Aalto-yliopiston teknillinen korkeakoulu
Elektroniikan, tietoliikenteen ja automaation tiedekunta
Tietoliikenne- ja tietoverkkotekniikan laitos

Diplomityö

Sovelluskehitysympäristön virtualisoinnin tuomat edut ja haitat

Heikki Salokanto

Helsingissä, 28.2.2010

Aihe hyväksytty koulutusneuvoston kokouksessa 15.2.2010.

Työn valvoja: **Prof. Jukka Manner**

Työn ohjaaja: **DI Pekka Pajuoja**

Tiivistelmä

Työn nimi Sovelluskehitysympäristön virtualisoinnin tuomat edut ja haitat		
Tekijä Heikki Salokanto	Tutkinto Diplomi-insinööri	Päivämäärä 28.2.2010
Professuuri S-38 Tietoverkkotekniikka	Kieli Suomi	Sivujen määrä 5 s + 76 s
Koulu ja tiedekunta Teknillinen korkeakoulu, Elektroniikan, tietoliikenteen ja automaation tiedekunta		
Laitos Tietoliikenne- ja tietoverkkotekniikan laitos (Comnet)		
Valvoja Professori Jukka Manner	Ohjaaja Diplomi-insinööri Pekka Pajuoja	

Tiivistelmä

Virtualisointi on ollut tekniikkana tunnettu jo kauan, mutta vasta viime vuosina virtualisointiratkaisuja on kehitetty niin pitkälle, että tekniikkaa voidaan hyödyntää lähes organisaatiossa kuin organisaatiossa. Tuotteisiin on lisätty monia uusia ominaisuuksia, yhteensopivuusongelmista on päästy pitkälti eroon ja tehokkuus on selvästi parantunut. Virtualisointi on jo käytössä monissa tuotantoympäristöissä.

Työssä perehdytään räätälöityjen, toimintakriittisten sovellusten kehitys-, testaus- ja tuotantoympäristön virtualisointiin. Tavoitteena on löytää uusia tapoja virtualisoinnin tuoman uuden abstraktiokerroksen hyödyntämiseen sovelluskehitysprosessissa, ja toisaalta tutkia, mitä haittoja ja riskejä tästä syntyy. Teoriaosuudessa perehdytään virtuaalikone- ja virtuaaliverkkoympäristön suunnitteluun sekä toimintavarmuuden, tietoturvan ja suorituskyvyn varmistamiseen suunnitellussa ympäristössä.

Use case -osuudessa tutkitaan Teknologian ja innovaatioiden kehittämiskeskuksen (Tekes) sovelluskehitysympäristöä erityisesti J2EE-sovellusten osalta. Sovelluskehitys siirretään virtualisoituun palvelinympäristöön, ja samalla otetaan käyttöön tekniikan mahdollistamia uusia toimintatapoja: mm. palvelimien kloonaus, palvelin-templatet ja *snapshotit*.

Toteutetun ympäristön jälkianalysoinnissa havaittiin hallinnan monipuolistuneen ja nopeutuneen selvästi, mutta suorituskykymittauksissa palvelimet eivät päässeet toivotulle tasolle. Jatkokehitysehdotuksina suositellaan sovelluspalvelinten klusterointia, reverse-proxyjä, muutoksia levyjärjestelmiin sekä ohjelmistojen vaihtamista vapaisiin lisenssikustannuksien säästämiseksi.

Avainsanat

virtualisointi, ohjelmistokehitys, ohjelmiston elinkaari, palvelinympäristö,
ketterä kehitys, ohjelmistotestaus

Abstract

Title Advantages and Disadvantages in Virtualization of Software Development Environment		
Author Heikki Salokanto	Level of studies Master's degree	Date February 28, 2010
Professorship S-38 Networking technology	Language Finnish	Page count 5 p + 76 p
School and faculty Aalto University School of Science and Technology Faculty of Electronics, Communications and Automation		
Department Department of Communications and Networking (Comnet)		
Supervisor Professor Jukka Manner	Instructor MSc Pekka Pajuoja	

Abstract

Virtualization is a long way from being a new invention, but it is only the last few years during which the technology has evolved into something truly useful—for almost any organization. Several handy new features have been introduced while at the same time the developers have gotten rid of most compatibility issues and significantly improved the performance. Consequently, virtualization is now in use in many a production environment.

This thesis is about virtualization of a software development environment of critical enterprise applications. The software development envinronment involves everything within the software's life-cycle: development, testing and production phases. The thesis aims to find new practices and conventions to exploit the new abstaction layer provided by virtualization to support the software development. Possible risks and disadvantages are analyzed and solutions presented. The theory part explains the planning of a virtualized server and network environment, paying attention especially to availability, security and performance.

The use case part then dissects the software and server environments of the Finnish Funding Agency for Technology and Innovation (Tekes). Tekes' software development environment of J2EE applications was virtualized in the course of writing this thesis, and new conventions suitable for this environment are being deployed. The most useful new services include server cloning, server templates and snapshots.

Post-analysis of Tekes' environement proved the new ways of management effective, but the performance test results did not quite satisfy the expectations. Suggestions for further development involve clustering of application servers, deployment of reverse-proxies and changes to storage systems. One of the easiest routes to cost savings is switching to free database and application server software.

Keywords

virtualization, virtualized environment, software development, software life-cycle, application servers, agile, software testing

Alkusanat

Kiitos mahdollisuudesta työn tekemiseen kuuluu sekä työnantajalleni HiQ Softplan Oy:lle että asiakasorganisaatiolle Teknologian ja innovaatioiden kehityskeskukselle (Tekes). Olen saanut molempien organisaatioiden työntekijöiltä arvokkaita neuvoja, vaatimusmäärittelyjä, kommentteja ja toivomuksia virtualisoidun ympäristön suhteen. Kiitos professori Jukka Mannerille sekä ohjaaja Pekka Pajuojalle hyvistä vihjeistä ja rakentavasta palautteesta.

Tekesin rahoituksenhallintajärjestelmästä voi lukea lisää aiemmin tehdyistä diplomitöistä: Peik Aschan, *Muutoksenhallinta tietojärjestelmän ylläpidossa*, Tampereen teknillinen yliopisto, 2002; sekä Samuli Lehto, *Performance Management in the J2EE-environment of Tekes*, Teknillinen korkeakoulu, 2005.

Heikki Salokanto

Helsingissä 28.2.2010

Sisältö

Tiivistelmä i

Abstract ii

Alkusanat iii

Sisällysluettelo iv

1 Johdanto 1
- 1.1 Työn päämäärä . . . 1
- 1.2 Työn laajuus . . . 1

2 Perinteisen järjestelmän ongelmat 3
- 2.1 Tausta . . . 3
- 2.2 Sovellukset ja sovelluspalvelimet . . . 4
  - 2.2.1 Sovellusten arkkitehtuuri . . . 5
  - 2.2.2 Vaatimukset . . . 6
  - 2.2.3 Sovellusten sijoittaminen sovelluspalvelimille . . . 7
- 2.3 Kehitettävää sovelluskehityksessä . . . 9
  - 2.3.1 Sovellusten siirtäminen ympäristöstä toiseen . . . 10
  - 2.3.2 Testausongelmat . . . 11
- 2.4 Suuren palvelinryppään aiheuttamia ongelmia . . . 11
- 2.5 Yhteenveto . . . 12

3 Virtuaalipalvelintekniikka 13
- 3.1 Virtuaalikoneen määritelmä . . . 13
- 3.2 Vaihtoehdot virtualisoinnille . . . 14
- 3.3 Virtualisointiarkkitehtuureja . . . 15
  - 3.3.1 Täysi virtualisointi . . . 16
  - 3.3.2 Paravirtualisointi . . . 18
  - 3.3.3 Emulointi . . . 19
- 3.4 Virtuaalipalvelinklusterit . . . 20
- 3.5 Muistin- ja levynkäyttö . . . 21
- 3.6 Virtuaalikoneiden hallinta . . . 22
- 3.7 Tunnettuja ongelmia . . . 24
- 3.8 Sopivan tuotteen valinta . . . 25
- 3.9 Yhteenveto . . . 28

4 Sovelluskehitysympäristön virtualisointi 29
- 4.1 Sovelluskehitysprosessi . . . 29
  - 4.1.1 Vesiputousmalli . . . 31
  - 4.1.2 Ketterät mallit . . . 31
- 4.2 Sovelluksen julkaisunhallinta . . . 33
- 4.3 Sovelluskehityksen ulkoistaminen virtualisoinnilla . . . 36
  - 4.3.1 Open Virtualization Format . . . 36

4.4 Verkon suunnittelu . . . 37
4.5 Palvelimien monitorointi . . . 39
4.5.1 Lista tarkkailtavista ominaisuuksista . . . 41
4.6 Varmuuskopiointi ja arkistointi . . . 42
4.6.1 Ajonaikainen varmuuskopio virtuaalikoneesta . . . 42
4.7 Sovellustestaus . . . 44
4.7.1 Sovellusten testaus työasemilla . . . 45
4.7.2 Suorituskykytestaus . . . 45
4.8 Yhteenveto . . . 47

5 Use case: Organisaation sovelluskehitys- ja palvelinympäristön virtualisointi 48
5.1 Järjestelmäarkkitehtuuri . . . 48
5.2 Palvelin- ja verkkoarkkitehtuuri . . . 48
5.3 Virtualisointiin ajavat tekijät . . . 49
5.4 Uuden ympäristön vaatimukset ja toivotut ominaisuudet . . . 50
5.4.1 Tehokkuus . . . 50
5.4.2 Skaalautuvuus . . . 51
5.4.3 Helppokäyttöisyys . . . 52
5.4.4 Ylläpidettävyys . . . 52
5.5 Tekesin virtuaaliympäristö . . . 52
5.6 Sovelluskehitysympäristö virtuaalisena . . . 53
5.7 Suorituskyvyn testaus . . . 55
5.7.1 Sovellustesti . . . 56
5.7.2 Prosessoriteho . . . 57
5.7.3 Levy-IO . . . 57
5.7.4 Testitulokset . . . 58

6 Saavutettujen etujen ja riskien analysointi 61
6.1 Uudet toimintatavat . . . 61
6.2 Lisenssiongelmat . . . 61
6.3 Suosituksia *use case* -organisaation ympäristön jatkokehitykseen . . . 62
6.3.1 Ratkaisuja suorituskykyongelmiin . . . 63
6.3.2 Ratkaisuja levytilaongelmiin . . . 63
6.3.3 Monitorointi ja statistiikan keruu . . . 64
6.3.4 Reverse-proxy . . . 64
6.3.5 WebLogic-palvelimien klusterointi . . . 65
6.4 Yhteenveto . . . 66

7 Loppusanat 67

Viitteet 68

Liite A 76

1 Johdanto

Virtualisointi on ollut tunnettu tekniikkana jo yli 30 vuotta. Yleiskäyttöiset, edulliset ratkaisut ovat kuitenkin tulleet varteenotettaviksi vaihtoehdoiksi lähes organisaatioon kuin organisaatioon vasta viime vuosina lukuisten kaupallisten ja vapaaseen ohjelmistoon perustuvien tuotteiden myötä. Tekniikka kehittyy edelleen erittäin nopeassa tahdissa, ja alustoihin esitellään uusia, käyttöä tehostavia ominaisuuksia lähes kuukausittain.

Virtuaalipalvelimia käytetään useimmiten vähentämään fyysisten palvelinkoneiden määrää ja helpottamaan palvelimien hallintaa. Virtualisoinnin avulla voidaan kuitenkin toteuttaa myös monikerroksisia järjestelmiä, joiden rakentaminen perinteisessä yksi palvelin – yksi kone -arkkitehtuurissa olisi erittäin työlästä: virtuaalikoneiden välille voidaan luoda esimerkiksi testausta varten virtuaaliverkkoja, jotka ovat vain rajoitetusti yhteydessä yrityksen tai yhteisön fyysiseen verkkoon. Näin sovellusten toiminta voidaan ensin testata turvallisesti eristetyssä verkkosegmentissä, ja sovellukset voidaan ottaa myöhemmin tuotantokäyttöön automatisoidusti siirtämällä kokonaisia virtuaalikoneita verkosta toiseen.

1.1 Työn päämäärä

Työssä tutkitaan virtualisoinnin tuomia mahdollisuuksia tehostaa ohjelmistokehitystä usean sovellustoimittajan ympäristössä. Varsinkin eri toimittajien sovelluksista ja usein myös sovellusten eri versioista koostuvan järjestelmän testaus asettaa palvelinympäristölle paljon haasteita, joita pyritään nyt ratkaisemaan virtualisoinnilla. Virtualisoidun kehitysympäristön käyttöönottoon liittyy myös testi- ja tuotantoverkkojen konfiguraatiot, palvelimien varmuuskopiointi, lisenssien hallinta, korkea saatavuus (*high availibility*) sekä riittävän suorituskyvyn varmistaminen. Työn ohessa evaluoidaan joitakin sovelluskehitysprosessia helpottavia työkaluja, kuten VMware Stage Manager.

Viime kädessä päämääränä on löytää ratkaisuja, jotka tuovat organsaatiolle kustannussäästöjä vähentyneen ylläpito- ja hallinointityön myötä sekä pienempänä fyysisten palvelinkoneiden määränä. Henkilöstön koulutustarve puolestaan lisäisi kustannuksia, joten ympäristö pitää pitää mahdollisimman helppokäyttöisenä tämän minimoimiseksi.

1.2 Työn laajuus

Tutkimus pätee ympäristöihin, joissa sovelluskehitystä tehdään ketterillä menetelmillä ja sovellukset ovat arkkitehtuuriltaan palvelin–asiakas-tyyppisiä. Osa työstä, erityisesti luku 3, on yleisempää taustatietoa, joka pätee kaikkiin virtuaaliympäristöihin.

Työssä tutkitaan Tekesin sovelluskehitys- ja verkkoratkaisuja use-case-periaatteella ja esitetään nimenomaan tähän ympäristöön sopivaa virtuaaliratkaisua sekä parannusehdotuksia nykyisiin toimintatapoihin. Ratkaisut ovat yleistettävissä samantyyppisiin organisaatioihin. Tekesin järjestelmillä tehdyt suorituskykymittaukset ovat suuntaa-antavia vastaavanlaisiin konfiguraatioihin, mutta sellaisenaan tarkkoja vain kyseisessä järjestelmässä.

Organisaation virtualisointiratkaisun osalta työssä käsitellään VMware ESX-palvelintuotetta. Lähes vastaavat ominaisuudet löytyvät kuitenkin – tosin eri nimillä – myös muiden valmistajien tuotteista. Linux-ratkaisut toimivat soveltaen kaikissa merkittävimmissä Linux-jakeluissa, vaikka työssä käsitelläänkin pääasiassa RedHat Enterprise Linuxia.

2 Perinteisen järjestelmän ongelmat

Tässä luvussa käsitellään vaatimuksia ja ongelmia, joita sovelluskehitys asettaa laitteisto- ja sovellusympäristölle. Ensin tutkitaan ympäristön edellytyksiä monikerroksisen sovellusarkkitehtuurin kannalta ja tämän jälkeen sovelluskehitysprosessin kannalta. Lopuksi käydään läpi fyysisten palvelimien aiheuttamia ongelmia. Työn kannalta luvun oleellisimpia asioita on, millaisiin ongelmiin sovelluspalvelinkonfiguraatioissa usein törmätään perinteisessä palvelinarkkitehtuurissa.

2.1 Tausta

Organisaatiot tarjoavat enenevässä määrin sekä asiakkailleen että työntekijöilleen erilaisia verkkopalveluja. Organisaation sisäisiä järjestelmiä voivat olla esimerkiksi toiminnanohjausjärjestelmät, projektin- tai prosessinhallinta, taloushallinnon tietojärjestelmät, dokumentinhallinta sekä intraweb, johon voi kuulua esimerkiksi organisaation uutiset, puhelinluettelo, työntekijöiden keskustelupalsta sekä erilaiset henkilöstöasiat. Asiakkaille tarjottavia palveluja ovat muun muassa tekninen tuki, verkkokauppa ja asiakkuudenhallinta. Yhteistyökumppanit voidaan jopa päästää joihinkin organisaation sisäisiin järjestelmiin. Kaikki nämä tietojärjestelmät ovat organisaation toiminnan kannalta kriittisiä, joten niitä pitää jatkuvasti ylläpitää ja useimpia myös jatkokehittää.

Harvalla – jos millään – organisaatiolla on resursseja toteuttaa kaikki tietojärjestelmät talon sisäisesti, joten lähes aina käytetään ulkopuolisia sovellustoimittajia. Osa sovelluksista voi olla vapaita, esimerkiksi GPL-lisensoituja[1] ohjelmistoja, osa valmiita kaupallisia tuotteita ja osa räätälöityjä ratkaisuja. Näin ollen sovellustoimittajia voi olla jopa kymmeniä, ja hankalimmaksi asiaksi tietojärjestelmien kehityksessä nouseekin sovellusten integroiminen yhteen.

Integraation olennaisimpia osia on integraatiotestaus – sovellusten testaaminen kokonaisuutena – joka asettaa kymmenien sovellusten tapauksessa suuria vaatimuksia testiympäristölle. Dynaamisen, tuotantoa vastaavan testiympäristön rakentaminen on erittäin haastavaa, ja merkittävän suuri osa testaukseen käytettävissä olevasta ajasta kuluukin palvelinympäristön rakentamiseen ja konfiguroimiseen [1].

Yrityksissä on alettu myös herätä fyysisten palvelimien tuottamiin konkreettisiin ongelmiin. Palvelinsali täyttyy, ja lisätilan rakentaminen ilmastointeineen, turvalaitteineen ja sähkönsaanteineen on kallista. Sähkönkulutus ja ekologia otetaan nykyisin huomioon: pienen yrityksen kymmenen palvelinta tuottavat jo vuosittain monen tuhannen euron sähkölaskun. Suurempi koneiden määrä tarkoittaa luonnollisesti aina suurempaa vikaantumistiheyttä, joka taas näkyy huolto- ja ylläpitokuluissa.

[1] GNU General Public License, `www.gnu.org/copyleft/gpl.html`

Perinteisen palvelinsalin yleinen ongelma on yksittäisten palvelimien alhainen käyttöaste – keskimäärin alle 15 % [2]. Joskus samalle palvelimelle voidaan asentaa eri palveluita, jolloin käyttöaste paranee, mutta ongelmaksi voi muodostua esim. tietyn palvelun hetkittäinen korkea kuormitus, tietoturva, stabiilius, käyttöjärjestelmävaatimukset tai palvelinohjelmiston vaatimat erikoiskonfiguraatiot.

2.2 Sovellukset ja sovelluspalvelimet

Sovelluspalvelimella tarkoitetaan ohjelmistoa, joka tarjoaa alustan tietyn kehyksen tai mallin mukaan rakennettujen sovellusten ajamiseen. Sovelluspalvelimet tarjoavat puitteet paitsi sovellusten suorittamiseen, myös niiden hallintaan, monitorointiin ja kehitykseen. Huomionarvoista on, että sovelluspalvelimista puhutaan ajoittain sekavasti tarkoittaen sekä palvelinohjelmistoja että palvelinlaitteistoja. Tämän vuoksi tässä työssä käytetään fyysisestä laitteesta puhuttaessa termiä "palvelinkone" tai "fyysinen palvelin".

Tunnetuimpia sovellusalustoja ovat Java Enterprise Edition [www-1] (Java EE), .Net Microsoft Application Platform [www-2] (MSAP), Customer Information Control System [www-3] (CICS) sekä Common Object Request Broker Architecture [www-4] (CORBA). Sovelluspalvelimia vastaavasti ovat mm. BEA/Oracle WebLogic [www-5], JBoss [www-6] ja Apache Tomcat [www-7] (Java EE -alusta), Microsoft Windows Server 2003 [www-8] ja 2008 [www-9] (MSAP-alusta) sekä IBM CICS Transaction Server [www-10].

Nykyään vahva suuntaus on selaimella käytettävät sovellukset eli web-sovellukset (*web applications*), joiden etuja sovelluskehityksen kannalta ovat muun muassa [3]:

- Sovelluslogiikan selkeä erottaminen käyttöliittymästä
- Sovelluskehittäjien roolien selkeyttäminen mm. edellä mainitun jaon perusteella
- Laaja valikoima eri sovellusrajapintoja (*application programming interface*, API) ja kehyksiä (*framework*)
- Keskitetty hallinta käytettävän kehyksen mukaan, esimerkiksi käyttäjänhallinta ja käyttöliittymäpohjat (*templates*)
- Kehitystyökalujen suuri määrä ja yhteensopivuus
- Testaustyökalujen monipuolisuus ja valmis integrointi rajapintoihin
- Kustannusten vähentäminen valmiita kehyksiä käyttämällä
- Valmiita tuotteita tarjolla, mikäli sovellus halutaan ostaa itse toteuttamisen asemesta

Web-sovellusten käytännön etuja asiakaspuolella (*client-side*) *ovat* helppo ja nopea käyttöönotto selainkäyttöliittymän ansiosta, keskitetyt päivitykset (käyttäjien työasemille ei tarvitse tehdä päivityksiä), parempi tietoturva (vähemmän työasemilla käsiteltävää dataa) sekä alustariippumattomuus.

2.2.1 Sovellusten arkkitehtuuri

Arkkitehtuurilla tarkoitetaan tässä tapaa, jolla sovelluksen eri funktionaaliset komponentit, kuten käyttöliittymä, businesslogiikka ja datavarasto, toimivat ja kommunikoivat keskenään.

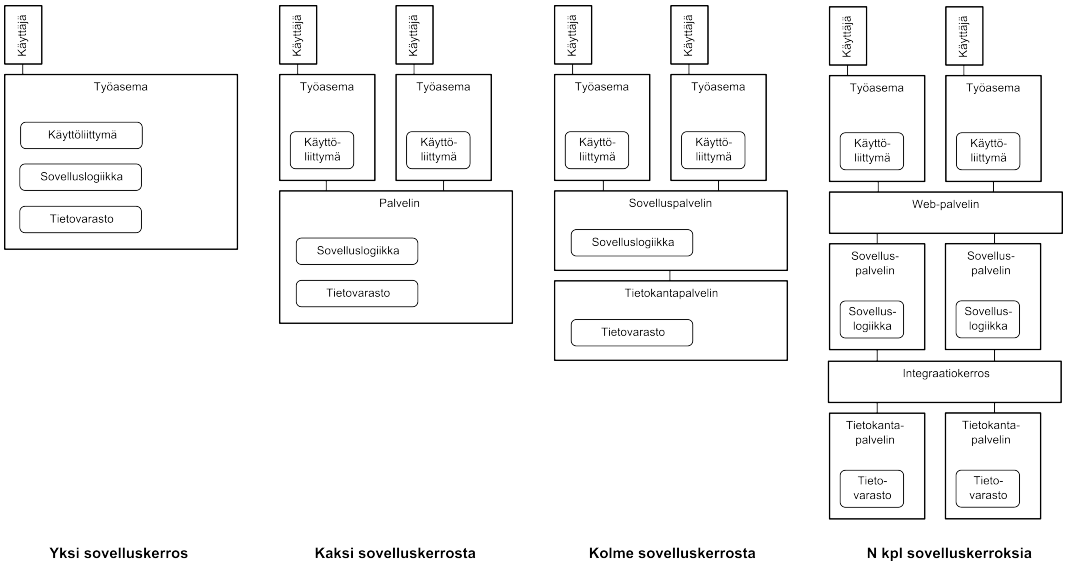
Yksinkertaisissa yhden käyttäjän sovelluksissa arkkitehtuuriin voi kuulua vain yksi sovelluskerros (kuva 1): sama ajettava ohjelma näyttää käyttöliittymän, suorittaa kaikki toiminnot ja tallentaa tiedot jollekin massamuistille. Tällaisia sovelluksia ovat pääsiassa työasemilla käytettävät ohjelmistot, kuten tekstinkäsittely.

Usean käyttäjän sovelluksissa tarvitaan vähintään kaksi sovelluskerrosta: asiakasohjelma sekä palvelinohjelma. Asiakasohjelma toimii käyttöliittymänä ja palvelinohjelma varastoi datan. Sovelluslogiikka voi olla sijoitettuna kumpaan tahansa ohjelmaan tai osittain molempiin. Esimerkki kahden sovelluskerroksen palvelusta on WWW-sivu, jossa WWW-palvelin tarjoaa WWW-sivun näytettäväksi käyttäjän selainohjelmalle. Sekä palvelin että selain saattavat suorittaa sivuun liittyvää logiikkaa.

Kaksi kerrosta ei riitä vielä erottamaan sovelluksen eri funktionaalisia komponentteja toisistaan. Niinpä usein käytetään kolmen kerroksen mallia, jossa datavarasto (tietokanta) erotetaan sovelluspalvelimesta. Suurin osa pienistä ja keskisuurista web-palveluista toteutetaan tällä mallilla.

Monimutkaiset web-sovelluskokonaisuudet tarvitsevat kuitenkin vielä parempaa skaalautuvuutta, joka saavutetaan lisäämällä kaksi kerrosta: esityskerros, joka hakee businesslogiikkaa suorittavilta palvelimilta tiedot ja yhdistää ne halutunlaiseksi web-sivuksi, sekä integraatiokerros, joka tarjoaa eri tietovarastojen tietoja keskitetysti eri sovelluspalvelimille [4]. Tätä kutsutaan yleisesti N-kerroksiseksi (*N-tier*) arkkitehtuuriksi.

Mitä enemmän kerroksia arkkitehtuuriin kuuluu, sitä skaalautuvampi ja joustavampi järjestelmästä on mahdollista rakentaa. Kerrosten sisäistä toimintaa pystytään muuttamaan ilman, että muihin kerroksiin joudutaan tekemään muutoksia; esimerkiksi tietokantaa voidaan vaihtaa tai sovelluspalvelin klusteroida kolmelle eri fyysiselle palvelimelle. Toisaalta kerrosten lisääminen muuttaa ympäristöä merkittävästi kompleksisemmaksi. Kerrosten väliset rajapinnat täytyy olla tarkoin määriteltyjä, palvelinkoneita tarvitaan useita, järjestelmän testaaminen muuttuu vaikeaksi, ja ylläpito vaatii paljon asiantuntemusta.



Kuva 1: Sovelluksen arkkitehtuuriratkaisut sovelluskerrosten määrän mukaan

Tässä työssä kaikki käsiteltävät sovellukset ovat lähtökohtaisesti JavaEE-alustalla toteutettuja N-kerroksisia web-sovelluksia, joissa käyttäjämäärät liikkuvat vähintään sadassa yhtäaikaisessa käyttäjässä. Nämä sovellukset ovat pääosin yrityksen sisäverkossa toimivia, mutta tarjoavat joitakin rajapintoja datan esittämiseen ulkoisille käyttäjille. Tietoturva on äärimmäisen kriittistä: yrityksen luottamuksellista tietoa ei saa vuotaa ulkoisiin rajapintoihin.

2.2.2 Vaatimukset

Käyttäjälle sovelluspalvelimen, niin laitteiston kuin ohjelmistonkin, tehokkuus ilmenee lyhyinä vasteaikoina ja virheettömänä toimintana. Sovelluskehittäjän kannalta taas palvelimen pitäisi olla helposti konfiguroitavissa ja ylläpidettävissä, ja ohjelmiston uuden version asentamisen pitäisi sujua huomaamattomasti ja mahdollisimman automatisoidusti. Sovelluksen omistajan (tilaaja, asiakas) intresseissä sen sijaan ovat mahdollisimman vähäiset käyttökatkokset niin virhetilanteista kuin huoltotöistäkin johtuen, nopea projektin läpivienti sekä alhaiset kokonaiskustannukset.

Sovelluksen arkkitehtuurin ja sen myötä palvelinympäristön valinnalla on suuri vaikutus vaatimusten toteutumiseen. Kymmenelle palvelimelle hajautettu järjestelmä voi tuottaa nopeat vasteajat ja hyvän skaalautumisen, mutta on hankalasti ylläpidettävä ja testattava sekä kallis toteuttaa. Sovelluskehitysympäristön rakentaminen monimutkaiselle sovellukselle on niin ikään työlästä – esimerkiksi testausta varten kaikkien sovelluskerrosten pitäisi olla toiminnassa myös testiympäristössä.

2.2.3 Sovellusten sijoittaminen sovelluspalvelimille

Kaikissa usean sovelluksen organisaatioympäristöissä joudutaan miettimään, kuinka monta sovelluspalvelinta asennetaan ja kuinka sovellukset jaetaan näille palvelimille. Kategorisoidaan sovellukset kahteen luokkaan niiden resurssivaatimusten perusteella:

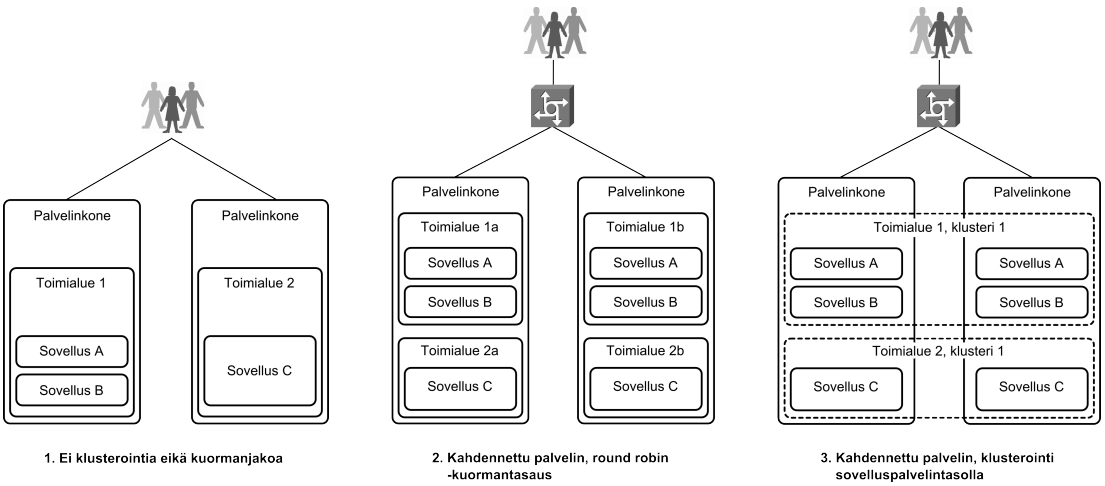
- Kevyet sovellukset, joita voidaan ajaa useita samalla palvelinkoneella
- Raskaat sovellukset, jotka voivat käyttää kerralla yli yhden palvelinkoneen resurssit

Sovelluksen resurssivaatimukset eivät välttämättä riipu itse sovelluksen monimutkaisuudesta, sillä monimutkainen sovellus voi pienillä käyttäjävolyymeillä toimia vähillä laiteresursseilla, ja vasta paljon käytettynä tarvitsee useita tehokkaita palvelimia. Toisaalta yksinkertainenkin sovellus voi yllättävän kuormituspiikin alla aiheuttaa palvelimen ylikuormittumisen.

Sovellusten käydessä raskaammiksi ja käyttäjämäärien kasvaessa vasteajat alkavat pidentyä, ja yhden palvelimen suorituskyky ei enää riitä pitämään käytettävyyttä vaaditulla tasolla. Sovelluskuorman jakaminen usealle palvelimelle voidaan toteuttaa usealla eri tavalla (kuva 2):

1. Jaetaan eri sovellukset eri palvelimille. Kevyitä sovelluksia voidaan edelleen ajaa useita samalla palvelimella, mutta raskaalle sovellukselle omistetaan koko palvelin. Jako on helppo toteuttaa ja konfiguroida, eikä käytännössä vaadi sovelluksiin muutoksia. Skaalautuvuus ja vikasietoisuus ovat kuitenkin edelleen yhtä heikkoja kuin yhden palvelimen tapauksessa.
2. Asennetaan toinen vastaavanlainen palvelin, joka ajaa kaikkia sovelluksia. Jaetaan käyttäjät vuorotellen kummallekin palvelimelle ulkoisella kuormantasaimella tai proxy-palvelimella. Ratkaisulla saavutetaan vikasietoisuus ja kuorman jakautuminen lähes tasan palvelimien kesken sekä uutena mahdollisuutena sovellusten päivittäminen palvelin kerrallaan ilman toimintakatkoksia [5]. Kaikki sovellukset eivät kuitenkaan toimi tällaisessa kuormantasauksessa, koska tilatietoa ei vaihdeta sovellusten kesken. Tämä voi johtaa siihen, että saman sovelluksen eri instanssit ovat eri tiloissa.
3. Sovelluspalvelimissa on mahdollisuus tehdä myös ohjelmistotason klusterointia, joka olellisesti tekee sovelluksista yhdessä toimivia ryppäitä, jotka vaihtavat keskenään tietoa ja käyttävät yhteisiä resursseja hallitusti. Vikasietoisuus on parempi kuin yksinkertaisella kuormantasaimella: käyttäjät eivät edes menetä istuntojaan, vaikka jokin komponentti vikaantuu [5]. Klusterointi vaatii kuitenkin muutoksia itse sovelluksiin ja on huomattavasti monimutkaisempi toteuttaa kuin vaihtoehdot

1 ja 2. Lisäksi klusterointiominaisuus kuuluu yleensä vain kalliimpiin lisensseihin (esim. Oracle WebLogic Suite [www-5]).



Kuva 2: Kuormantasaus ja klusterointi

Koska tietyn toimialueen[2] (*domain*) puitteissa toimiva klusteri voi käsittää useita palvelinkoneita, täytyy samalla miettiä sovellusten jako toimialueisiin ja palvelininstansseihin [6]. Vaihtoehtoisia tapoja toteuttaa jako ovat:

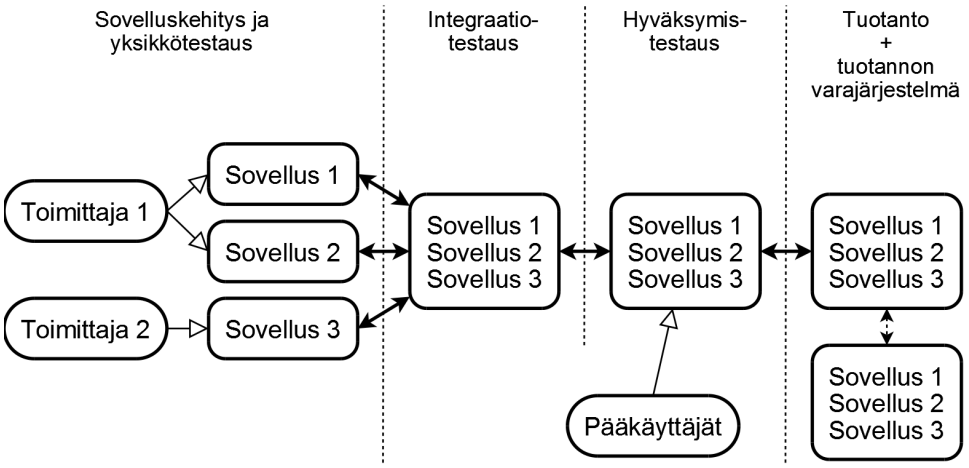
- Sovellusten ajaminen samassa sovelluspalvelininstanssissa. Tällöin ne näkyvät järjestelmälle yhtenä Java-prosessina, ja ongelmatilanteissa kaikki instanssissa ajettavat sovellukset kuolevat tai joudutaan lopettamaan. Konfigurointi on helppoa ja resursseja säästyy. Vikasietoisuus on heikko.
- Sovellusten ajaminen samassa toimialueessa, mutta eri palvelininstansseissa. Resursseja kuluu enemmän kuin yhden palvelininstanssin tapauksessa, mutta hallittavuus ja vikasietoisuus paranee, koska instansseja voidaan käynnistää ja sammuttaa yksitellen. Eri instansseille voidaan kohdistaa eri asetuksia, mikä sekä monipuolistaa että monimutkaistaa hallintaa.
- Sovellusten ajaminen eri toimialueissa. Sovellukset ovat tällöin eristettyinä toisistaan, mikä lisää tietoturvaa. Hallittavuus kuitenkin hankaloituu, koska eri toimialueet konfiguroidaan täysin toisistaan erillään, ja resursseja tarvitaan enemmän.

Sovellusten määrän ja vaatimusten muuttuessa palvelinkone-, toimialue-, klusteri- ja instanssijakoa joudutaan muuttamaan. Muutosta vaikeuttaa eri ohjelmistojen, käyttöjärjestelmien ja arkkitehtuurien yhteensopimattomuus. Pahimmillaan jopa saman sovelluspalvelintuotteen peräkkäiset versiot ovat keskenään yhteensopimattomia.

[2] BEA/Oracle WebLogic -palvelimen hallinnallinen yksikkö, johon voi kuulua useita klustereita ja palvelininstansseja.

2.3 Kehitettävää sovelluskehityksessä

Sovelluskehitystä varten tarvitaan lähtökohtaisesti kehitys-, testaus- ja tuotantoympäristöt. Tilanne on yksinkertaistettuna kuvan 3 mukainen. Sovelluskehitysprosessi on tarkemmin kuvattu luvussa 4.1.



Kuva 3: Sovellusten kehitysprosessi

Kehitysympäristössä sovelluskehittäjät suunnittelevat, ohjelmoivat ja suorittavat yksikkötestejä. Tarkoitukseen sopii esimerkiksi Windows-työasema Eclipsellä [www-11] ja WebLogic-palvelimella varustettuna. Joskus kehitysympäristössä voidaan ajaa myös taustapalveluja, kuten tietokannan kevytversiota. Pelkästään kehitysympäristölle ominaisia työkaluja ovat mm. virheiden raportointi, versionhallinta sekä dokumentaationhallinta. Ympäristö eroaa merkittävästi tuotantoympäristöstä niin laitteiston kuin ohjelmistonkin osalta.

Testausympäristö kattaa sovellusten monivaiheisen testauksen. Testaus voi kulkea esimerkiksi seitsenportaisen mallin mukaan: yksikkötestaus → integraatiotestaus → järjestelmätestaus → järjestelmien integraatiotestaus, sekä ennen käyttöönottoa alphatestaus → betatestaus → hyväksymistestaus. Tämä perustuu ohjelmiston kehittämiseen vesiputousmallin (kpl. 4.1.1) mukaan ja testaamiseen ns. V-mallin mukaan – tarkoittaen, että jokaiselle kehitysvaiheelle on olemassa myös vastaava testausvaihe [7]. Kettärässä kehityksessä (kpl. 4.1.2) testausvaihdeiden määrä voi olla supistettu jopa vain kolmeen:

- Yksikkötestaus: suoritetaan sovelluskehittäjien omissa ympäristöissä
- Integraatiotestaus: testataan sovelluksen toimivuus sekä taustamoduulien että rinnakkaisten sovellusten kanssa
- Hyväksymistestaus: testataan koko järjestelmän toiminta tuotantoa vastaavassa ympäristössä

Testausvaiheiden vähentäminen keventää prosessia oleellisesti, mutta toisaalta testausta suoritetaan koko kehityksen ajan kaiken muun aktiviteetin rinnalla, joten sen pitää olla erittäin sulavaa ja mahdollisimman automatisoitua. Tämä puolestaan asettaa suuria vaatimuksia testausympäristölle sekä automaatioratkaisuille (esimerkiksi CruiseControl [www-12]). Testauksen sujuvuus on tämän työn kannalta kehitysprosessin merkittävimpiä asioita.

Sovellusten siirtoa ympäristöstä toiseen sekä testausta yleensä vaikeuttaa ympäristöjen erilaisuus, joka johtuu resurssien rajallisuudesta ja nopeasti muuttuvista tarpeista. Testausympäristöihin ei pystytä hankkimaan niin kallista laitteistoa kuin tuotantoon: esimerkiksi kuormantasaajaa ei monesti asennetta testaukseen ollenkaan. Lisäksi samalla testipalvelimella haluttaisiin voida testata useita sovelluksia, jotka tuotannossa toimivat eri palvelimilla. Palvelimien konfiguraatiohallinta pitää koordinoida tarkoin, jotta tuotanto- ja testiympäristöt eivät hiljalleen muutu erilaisiksi. Esimerkiksi käyttöjärjestelmäpäivitysten asentaminen testiympäristöihin, mutta ei tuotantoon, aiheuttaa jo kriittisen eroavuuden.

Lisäksi kokonaisuuden hallintaa monimutkaistavat tietokannat ja muut sovellukseen integroitavat organisaation keskitetyt tietojärjestelmät, kuten dokumentinhallinta, julkaisujärjestelmä ja muut ulkoiset palvelut. Sovelluksen täydellinen toiminta edellyttäisi kaikkien integroitujen komponenttien läsnäoloa, mutta testiympäristöön ei yleensä pystytä järjestämään kuin osa niistä. Osa taustapalveluista mahdollisesti korvataan nk. jäljitelmillä (*mock*), jotka imitoivat oikeita järjestelmiä palauttamalla jonkin staattisen vastauksen. Kompromissiratkaisut kuitenkin johtavat siihen, että testiympäristö eroaa yhä enemmän tuotannosta, ja näin ollen käy todennäköiseksi, että tuotannossa tulee eteen ongelmia, joita testissä ei voida havaita.

2.3.1 Sovellusten siirtäminen ympäristöstä toiseen

Perinteisesti räätälöidyistä sovelluksista on pitänyt kääntää eri versiot eri ympäristöjä varten. Prosessi on hidas ja työläs, koska tietyn järjestelmäkokonaisuuden kaikkien eri sovellustoimittajien sovelluksista pitää saada versiot kutakin ympäristöä varten. Sovelluspalvelimien toimialueita, klustereita ja palvelininstansseja ei voida myöskään suoraan kopioida eri ympäristöihin, koska asetuksissa on määritelty muun muassa käytettävät tietokannat, salasanat ja tunnistautumiskäytännöt ympäristökohtaisiksi [6]. Pahimmillaan sovelluksen käännösvaiheessa on jo sidottu asioita kiinteisiin IP-osoitteisiin ja tietokantojen nimiin.

Selvä tarve on siis kehittää yleinen ratkaisu, miten ympäristöjä voidaan helposti, jopa automatisoidusti, siirtää palvelimelta toiselle, verkosta toiseen ja testiympäristöstä tuotantoon ilman, että sovellusta joudutaan kääntämään uudestaan.

2.3.2 Testausongelmat

Jatkuvasti kehitettävillä sovelluksilla testausta halutaan usein tehdä kolmessakin vaiheessa yhtä aikaa: sovelluskehittäjät testaavat tuoreinta kehitysversiota eristetyssä ympäristössä, eri toimittajien sovelluksia testataan yhdessä integraatioympäristössä, ja pääkäyttäjät testaavat tuotantoon vietäviä sovelluksia hyväksymistestausympäristössä. Vesiputousmallia käytettäessä testausvaiheita on vielä enemmän.

Testausympäristöt eivät vaadi merkittäviä laitteistoresursseja vähäisten käyttäjämäärien vuoksi, mutta testauksen sujuvuuden ja mielekkyyden kannalta ympäristöille asetetaan tiettyjä kriteereitä. Erityisesti niiden hallinnan pitää olla joustavaa ja nopeaa: ei ole mielekästä, että suuri osa testausajasta kuluu ympäristön konfiguroimiseen. Funktionaalisten testien kannalta vasteajoilla tai maksimikäyttäjävolyymillä ei ole merkitystä, vaan testejä voidaan suorittaa selvästi tuotantoa heikompitehoisilla palvelimilla [8].

Sen sijaan suorituskykytestaus on erittäin herkkä vallitsevalle ympäristölle. Jos halutaan mitata tuotantoympäristön suorituskykyä, testauksen pitää tapahtua täsmälleen tuotannon kaltaisessa ympäristössä. Lineaariset arviot kuorman kasvamisesta käyttäjämäärän mukaan antavat väärän kuvan ympäristön suorituskyvystä [8]. Lisäksi pullonkaulat saattavat vaihtua kuormituksen kasvaessa, jolloin arviot eivät osu lähellekään todellista tilannetta. Organisaatioilla ei kuitenkaan usein ole resursseja hankkia pelkästään suorituskykytestausta varten toista tuotantojärjestelmää, joten testaus pitää voida suorittaa käytettävissä olevilla laitteilla.

2.4 Suuren palvelinryppään aiheuttamia ongelmia

Palvelinkoneiden nopeasti kasvava määrä tuottaa suuren joukon ongelmia. Monissa organisaatioissa tulee vastaan tilaongelma: palvelimet pitää sijoittaa asianmukaisesti ilmastoituun, tietoturvalliseen ja paloturvalliseen tilaan, jossa on riittävä sähkönsyöttökapasiteetti, tietoliikenneyhteydet sekä lukittavat räkkikaapit kaikille laitteille. Alun perin muutamalle palvelimelle suunniteltu tila on monesti jo täynnä, ja uusille koneille ei yksinkertaisesti ole paikkaa. Uuden palvelinhuoneen rakentaminen on kallista ja vaatii suuren remontin.

Koneiden sähkönkulutus aiheuttaa haasteita paitsi palvelinhuoneen suunnittelun, myös suoraan käyttökustannusten muodossa. Yhden palvelinkoneen keskimääräiseksi tehontarpeeksi voidaan arvioida 400 W, mikä tarkoittaa keskisuuren yrityksen maksamalla 0,06 €/kWh –hinnalla sähkölaskua 216 €/vuosi. Tämän lisäksi täytyy laskea ilmastoinnin, varavirtajärjestelmän (*uninterruptible power supply*, UPS) ja muiden lisälaitteiden kulutus, joka on karkeasti arvioiden kaksi kertaa palvelimen oman tehontarpeen verran. Näin ollen sähkön kokonaisvuosihinnaksi tulee n. 648 €/kone. Tällä hetkellä palvelin käyttää siis hankintahintansa verran sähköä n. 3–5 vuodessa, mutta sähkön hinnan nousun myötä aika voi lyhentyä huomattavastikin [2].

Suuren palvelinjoukon ylläpito ja hallinta pitää organisoida tehokkaasti. Rutiinitoimenpiteiden hoitaminen, palvelinten tarkkailu ja varmuuskopiointi vaativat keskitettyjä järjestelmiä, joiden asentaminen esimerkiksi aina uuden palvelimen käyttöönoton yhteydessä on helposti muutaman henkilötyöpäivän lisäurakka IT-tiimille.

Useimmat yritykset pitävät kuitenkin suurimpana ongelmana palvelininfrastruktuurin hintaa [2]. Yrityksen koosta ja palvelintarpeesta riippuen hinta voi joskus ollakin kriittinen kysymys, mutta todellisuudessa esimerkiksi sovelluspalvelimeksi hankittava perustason palvelin on hyvin pieni osa IT-kokonaiskustannuksista. Sen sijaan uuden palvelimen hankkimisen vaatimien päätösten hyväksymisprosessi on suurissa organisaatioissa usein pitkä ja hidastempoinen, eikä uutta palvelinta saada käyttöön läheskään niin nopeasti kuin sille olisi tarvetta. Tämä on suuri ongelma nopeasti esiin tulevien testi- ja evaluointitarpeiden kanssa, sekä myös tuotantokäytössä tilanteessa, jossa resurssien loppuminen huomataan vasta, kun on jo liian myöhäistä.

2.5 Yhteenveto

Organisaatioiden prosessien- ja informaationhallinta on nykyään rakennettu suurelta osin eri tarkoituksiin räätälöityjen sovellusten varaan. Nämä sovellukset tarvitsevat palvelinjärjestelmiä, joiden tarpeet ovat kasvaneet räjähdysmäiseseti sovellusten määrän myötä. Uusien palvelimien hankkiminen ja ylläpitäminen aiheuttaa fyysisiä, rahallisia ja hallinnollisia ongelmia, joita perinteisellä “yksi sovellus yhtä palvelinkonetta kohden” -periaatteella ei pystytä järkevästi ratkaisemaan.

Sovellusten kasvava määrä aiheuttaa luonnollisesti myös painetta sovelluskehitykselle. Monen sovelluksen yhtäaikainen kehitys vaatii kullekin sovellukselle oman kehitysympäristönsä. Koska yhdessä kehitysympäristössä on useimmiten muutamia palvelimia, kehitysympäristöjä on yhtä sovellusta kohden muutamia ja kehitettäviä sovelluksia esimerkiksi viisi, pelkästään sovelluskehitykseen tarvittavien palvelinkoneiden määrä nousee helposti moniin kymmeniin. Tällaisen konemäärän hallinta vaatii organisoitua ja automatisoitua palvelimien hallintaa: koneiden asentamiseen, ylläpitoon ja tarkkailuun pitää olla selkeästi määritellyt, tehokkaat prosessit.

3 Virtuaalipalvelintekniikka

Tässä luvussa selvitetään virtuaalikoneisiin liittyvät yleiset peruasiat ja -teoriat. Luvussa keskitytään enemmän organisaatiotason virtualisointiratkaisujen, joilla voidaan hallita vähintään kymmeniä virtuaalikoneita, kuin yksittäisten virtuaalikoneiden tekniikkaan. Peruskäsitteiden ja -tekniikoiden lisäksi käsitellään tämän hetken merkittävimmät ja sovelluskehityskäytön kannalta mielenkiintoisimmat valmistajakohtaiset erityisratkaisut.

Oleellisimpia asioita luvussa on tunnistaa eri virtualisointitekniikoiden, erityisesti täyden virtualisoinnin ja paravirtualisoinnin, erot sekä sopivat käyttökohteet. Sovelluskehityksen kannalta virtuaalikoneiden hallintaominaisuudet ovat keskeisiä, koska palvelimien konfiguraatiota joudutaan muuttamaan usein.

3.1 Virtuaalikoneen määritelmä

Popekin ja Goldbergin vuonna 1974 laatiman määritelmän mukaan virtuaalikone on "tehokas, eristetty kopio oikeasta koneesta" [9]. Tarkemmin Popek ja Goldberg määrittelevät virtuaalikoneen toiminnan *Virtual Machine Monitorin* (VMM, myös *hypervisor*) avulla. VMM toimii kehyksenä kaikille virtuaalikoneille, ja sen pitää täyttää seuraavat kriteerit:

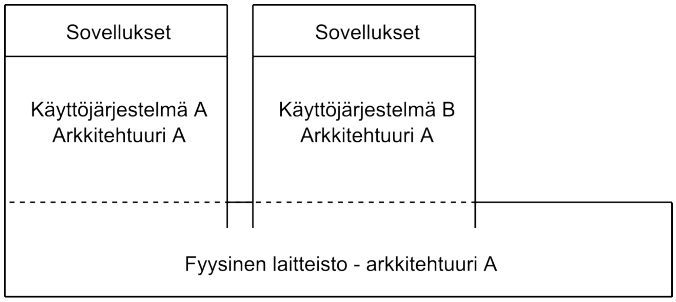
1. VMM tarjoaa ohjelmistolle ympäristön, joka on oleellisesti samanlainen kuin alkuperäinen laitteisto.
2. Virtuaaliympäristössä ajettavat ohjelmat kokevat korkeintaan hieman huonontuneen suorituskyvyn alkuperäiseen nähden. Käytännössä tämä tarkoittaa, että merkittävän suuri osa virtuaalikoneessa ajettavista konekäskyistä (*instructions*) pitää suorittaa ilman VMM:n puuttumista asiaan.
3. VMM hallitsee täysin virtuaalikoneiden käyttämiä järjestelmäresursseja.

Käytännössä Popekin ja Goldbergin kriteerit eivät aina toteudu täysin (luku 3.3.1). Jos ympäristö poikkeaa merkittävästi kriteereistä, aiheutuu tästä yllättäviä ja hankalasti ratkaistavia ongelmia esimerkiksi sovellusten vakauden ja tietoturvan suhteen: esimerkiksi, jos vastoin edellä mainittua kohtaa kolme virtuaalikoneesta käsin pääseekin hallitsemaan jotain fyysisen koneen resurssia, voi murtautuja päästä tätä kautta kiinni myös muiden virtuaalikoneiden tietoihin. Haittaohjelmien kirjoittajat hyödyntävät virtuaalikoneiden ja oikeiden koneiden eroja tekemällä ohjelmiinsa tunnistusmekanismeja, joiden avulla virtuaalikoneessa toimiessaan ohjelma voi esimerkiksi piilottaa osan toiminnallisuuttaan tietoturvaorganisaatioiden tutkijoita hämätäkseen, koska haittaohjelmien tutkiminen tehdään useimmiten juuri virtuaalikoneissa [10].

3.2 Vaihtoehdot virtualisoinnille

Virtualisointi ei ole aina tehokkain ja käytännöllisin tapa jakaa palvelinkoneen resursseja eri palveluille. Merkittävimmillä palvelinvalmistajilla kuten IBM, Hewlett-Packard ja Sun Microsystems, on monenlaisia suuriin palvelimiin kehitettyjä resurssienjakamisratkaisuja.

Yksinkertaisimmillaan palvelinresurssit saadaan tehokkaaseen käyttöön ajamalla useita sovelluksia rinnakkain samassa käyttöjärjestelmässä. Kun käytetään esimerkiksi WebLogic-toimialueita (luku 2.2.3), voidaan sovellukset jossain määrin eristää toisistaan: sovelluspalvelimella määritellään esimerkiksi sovellusten käyttäjätilit, jolloin käyttöjärjestelmätason käyttäjäprofiilit eivät vaikuta sovellusten toimintaan. Palvelinkone on kuitenkin vain yksi kone yhdellä käyttöjärjestelmällä, jolloin kaikki järjestelmän asetukset, kuten ohjelmistoversiot ja tietoturvakonfiguraatio, koskevat kaikkia ajettavia sovelluksia. Järjestely sopii siis tilanteisiin, joissa sovellukset ovat riittävässä määrin samankaltaisia toimiakseen samoilla käyttöjärjestelmän tarjoamilla palveluilla ja asetuksilla, eivätkä häiritse toistensa toimintaa.



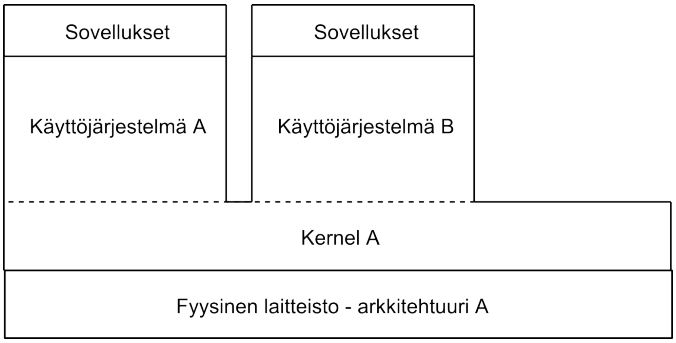
Kuva 4: Partitiointi laitteistotasolla

Suurissa päätietokoneissa (*mainframe*) on jo noin 20 vuoden ajan käytetty resurssien jakamista laitteistotasolla[3]. Tällä tavalla jaettuina osiot (*partitio,* engl. *partition*) ovat täysin eristettyjä ja itsenäisiä. Ensimmäisissä partitiointitoteutuksissa prosessorikapasiteetin jakaminen perustui fyysisten prosessorien osoittamiseen partitioille: partitioita ei voinut olla enempää kuin prosessoreja. Myöhemmin otettiin käyttöön tekniikoita, joissa esimerkiksi sadan millisekunnin jaolla voitiin jakaa prosessoriaikaa samalta prosessorilta: esimerkiksi yhdelle partitioille 20 ms, toiselle 30 ms ja kolmannalle 50 ms. Edelleen jako oli kuitenkin karkea ja staattinen.

Modernit tekniikat, kuten IBM POWER5 Micro-Partitioning, sallivat resurssien jakamisen kohtuullisen vapaasti: prosessoreja käsitellään virtuaalisina yksikköinä, joiden voidaan sallia ylittävän määritetyt partitiokohtaiset rajat, jos kapasiteettia on vapaana [11]. Laitteistotasolla tapahtuva partitiointi ei altista partitioissa toimivia järjestelmiä

[3] IBM Logical Partitioning (LPAR) julkaistiin syyskuussa 1990 ESA/390-järjestelmässä [www-13]. Tekniikkaa oltiin kehitetty 1980-luvun puolivälistä alkaen.

virtualisoinnin heikkouksille. Suurin ongelma ratkaisun käyttöönotossa useimmissa organisaatioissa on kuitenkin mainframe-tuotteiden korkea hinta. Esimerkiksi IBM:n mukaan jopa 1500 x86-palvelinta vastaavan IBM z10 Enterprise Class -päätietokoneen hinta alkaa noin miljoonasta US-dollarista [12].



Kuva 5: Partitiointi kernel-tasolla

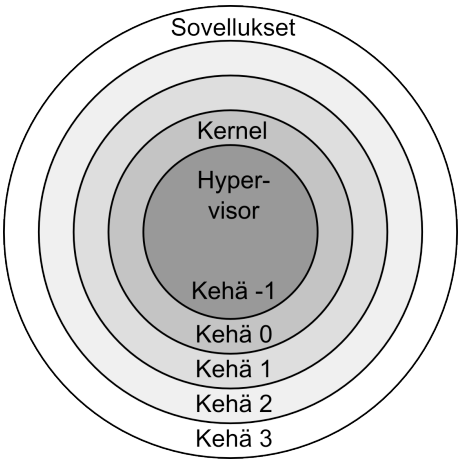
Sun Microsystems on vienyt partitioinnin lähemmäs virtualisointia kehittämällä Solarikseen käyttöjärjestelmäydintasolla (*kernel*-tasolla) tapahtuvan partitioinnin, jolla järjestelmä jaetaan loogisiin alueisiin (*zone*) [13]. Suurin ero virtualisointiin on, että tekniikalla ei pyritä virtualisoimaan laitteistoa vaan käyttöjärjestelmä. Käytännössä tämä tarkoittaa käyttöjärjestelmän jakamista itsenäisiin osakäyttöjärjestelmiin, jotka kuitenkin käyttävät samaa järjestelmäydintä ja jakavat samat fyysiset laitteistoresurssit ilman virtualisointia (kuva 5).

Partitiointi yhteistä ydintä käyttäen mahdollistaa hyvin kevyet osajärjestelmät. Näin ollen muistin käyttö on tehokkaampaa kuin itsenäisissä virtualisoiduissa koneissa, ja lisäksi ratkaisun tuoma prosessorilisäkuorma (*overhead*) on mitätön. Haittapuolena luonnollisesti seuraa, että kaikissa osajärjestelmissä tulee käyttää samaa käyttöjärjestelmäversiota. Vastaava tekniikka on käytössä Linux vServer -ratkaisussa [14].

3.3 Virtualisointiarkkitehtuureja

Prosessorit voivat toimia eri tiloissa sen mukaan, mikä käskyjä niiden sallitaan suorittaa. Täysin rajoittamatonta tilaa, jossa voidaan suorittaa mitä tahansa käskyjä ja osoittaa muistia mistä tahansa osoitteesta, kutsutaan kernel-tilaksi (*kernel-mode*, myös *master-mode*, *privileged mode* tai *supervisor mode*). Vastaavasti rajoitettu tila, jossa esim. muistin osoittaminen on rajoitettu tiettyyn muistialueeseen, on käyttäjätila (*user-mode* tai *unprivileged mode*). Tilat jaetaan turvallisuustason mukaan kehiksi käyttöjärjestelmäytimen ympärille: x86-prosessoreilla kehällä kolme (*Ring 3*) on käyttäjätila, kehät kaksi ja yksi ovat pääsääntöisesti käyttämättömiä ja sisin kehä (*Ring 0*) on kernel-tila (kuva 6) [15].

Virtuaaliympäristöissä on oleellista, että virtuaalikoneet eivät pääse käsiksi toisten virtuaalikoneiden muistiin, eivätkä esimerkiksi pysty suorittamaan VMM:a häiritseviä käs-



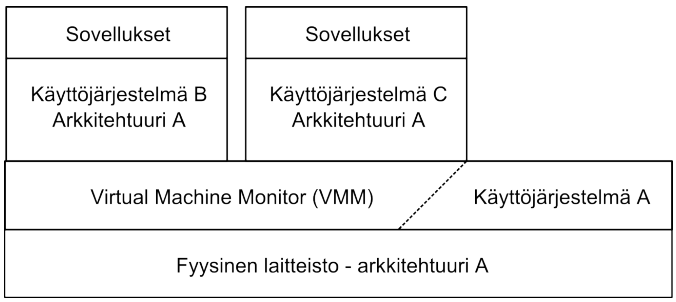
Kuva 6: x86-prosessorin kehäarkkitehtuuri virtualisointia käytettäessä. Ilman virtualisointia aluetta -1 ei ole.

kyjä. Jotta virtuaalikoneet voisivat kuitenkin suorittaa käskyjä natiivisti kernel-tilassa (kpl 3.3.1), lisäsivät Intel ja AMD x86-prosessoreihinsa ns. hypervisor-tilan, josta käytetään nimitystä "kehä -1" (kuva 6) [16].

3.3.1 Täysi virtualisointi

Virtualisointitekniikat kategorisoidaan sen mukaan, miten virtuaalikoneiden ajamat käskyt ja muistiosoitukset käsitellään. Pääkategoriat ovat täysi eli natiivi virtualisointi, paravirtualisointi ja emulointi.

Lähtökohtana täydessä virtualisoinnissa (*full virtualization*) on laitteiston täydellinen simuloiminen niin, että isäntäjärjestelmän päällä ajettavat virtuaalijärjestelmät käyttäytyvät aivan kuin niitä ajettaisiin oikealla laitteistolla. Näin ollen kaikkia käyttöjärjestelmiä, joita voidaan ajaa tietyllä prosessoriarkkitehtuurilla fyysisessä koneessa, voidaan ajaa myös kyseisellä laitteistolla toimivan täyden virtualisoinnin VMM:n alaisuudessa (kuva 7).



Kuva 7: Täysi virtualisointi

Popek ja Goldberg kiteyttävät täyden virtualisoinnin laitteistoarkkitehtuurimääritelmän seuraavasti:

> "For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions." [9]

Järjestelmän toiminnan kannalta herkillä käskyillä (*sensitive instructions*) tarkoitetaan kaikkia järjestelmän resurssien käyttöön vaikuttavia käskyjä, ja etuoikeutetuilla käskyillä (*privileged instructions*) sellaisia, jotka keskeyttävät suorituksen, jos toimitaan käyttäjätilassa, mutta eivät keskeytä, jos toimitaan kernel-tilassa. Vaatimus voidaan yksinkertaistaa muotoon: kaikki käskyt, jotka voivat vaikuttaa VMM:n toimintaan, keskeyttävät ja siirtävät suorituksen aina VMM:lle [9].

Näin ollen tutulla x86-arkkitehtuurilla ei ole voitu toteuttaa täyttä virtualisointia kuin vasta viimeaikoina laitteistopohjaisen virtualisoinnin tuen myötä, koska IA-32-käskykantaan kuuluu 17 toiminnan kannalta herkkää käskyä, jotka voidaan ajaa käyttäjätilassa. Rajoitus pystytään kiertämään muokkaamalla ohjelmakoodia nk. biinaarimuunnostekniikalla (*binary translation*) ennen sen suorittamista, jolloin käyttäjän kokema virtualisointi vastaa täysin täyttä virtualisointia [17]. Tämä kuitenkin huonontaa hyötysuhdetta eli lisää virtualisoinnin tehonhukkaa merkittävästi.

Useat valmistajat ovatkin sisällyttäneet suorittimiinsa virtualisointia tukevia tiloja ja ominaisuuksia, joilla saadaan täysi virtualisointi käyttöön ilman ohjelmistopohjaisia kiertoratkaisuja. Näistä merkittävimmät ovat:

- Intel Virtualization Technology (VT) x86-prosessoreille [17]
- AMD Pacifica (AMD-V) x86-prosessoreille [www-14]
- IBM Advanced POWER [www-15]
- Hitachi Virtage (BladeSymphony-palvelimille) [www-16]
- Sun Microsystems "Hyper-privileged execution mode" UltraSPARC T1 ja T2 -prosessoreille[4] [www-17]

Näistä etenkin uusi x86-prosessorien virtualisointituki on merkittävästi lisännyt edullisten virtuaaliratkaisujen houkuttelevuutta. Suorituskykyisen palvelimen, joka pystyy virtualisoituna ajamaan jopa kymmeniä yrityksen vanhoja palveluita, saa nyt pitkälti alle 5000 eurolla[5], kun ennen virtualisointia tällaisesta olisi joutunut maksamaan helposti yli kymmenkertaisen summan käyttäen esim. IBM:n tai Sunin laitteistopohjaisia ratkaisuja.

[4] Sun on julkaissut nämä prosessorit vapaana koodina

[5] Esimerkiksi Dell PowerEdge ja HP ProLiant -palvelimet alkaen n. 2000 €

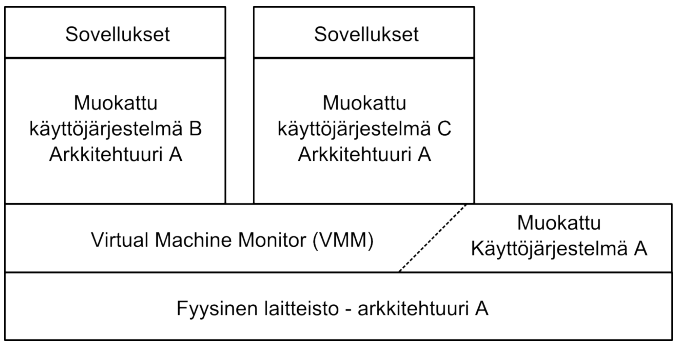
Täyden virtualisoinnin alustaratkaisuja x86-arkkitehtuurille ovat mm.:

- WMware ESX [www-18] ja VMware Server [www-19]
- Microsoft Virtual Server [www-20] ja Hyper-V [www-21]
- Sun VirtualBox [www-22] (osa Sun Microsystemsin xVM-tuoteperhettä [www-23])
- Kernel-based Virtual Machine [www-24] (KVM)

Muihin virtualisointitekniikoihin nähden täyden virtualisoinnin suurimpia etuja on, että vierasjärjestelmää (*guest operating system*) ei tarvitse mitenkään muokata virtualisointiin sopivaksi, vaan virtuaaliympäristössä voidaan ajaa mitä tahansa kyseiseen prosessoriarkkitehtuuriin tarkoitettua käyttöjärjestelmää sellaisenaan. Toisaalta täysi virtualisointi vaatii laitteistotukea ja puhtaasti toteutettuna generoi erittäin paljon keskeytyksiä VMM:lle, mikä hidastaa toimintaa [19].

3.3.2 Paravirtualisointi

Paravirtualisoinnissa virtuaalikoneille tarjottava rajapinta ei vastaa täysin oikeaa laitteistoa. Tämän ansiosta VMM voidaan rakentaa yksinkertaisemmaksi ja tehokkaammaksi, mutta tekniikka vaatii muutoksia ajettaviin virtuaalikoneisiin – toisin sanoen mitä tahansa käyttöjärjestelmää ei voida ajaa paravirtualisoidussa ympäristössä (kuva 8). Teknikka perustuu IBM:n jo vuonna 1972 kehittämään VM/370 Time-sharing System -järjestelmään, joka tarjosi sen käyttäjille erillisiltä ja itsenäisiltä näyttävät System/370-järjestelmät [20].



Kuva 8: Paravirtualisointi

Paravirtualisoinnin hyödyistä ja haitoista ollaan jokseenkin eri mieltä. Virtualisointiratkaisu Xen [www-25] esittää tekniikan tarjoavan "erittäin suurta suorituskykyä ja jopa kymmenen kertaa pienemmän lisäkuorman kuin perinteiset virtualisointiratkaisut" (tarkoittaen täyttä virtualisointia). Lisäksi paravirtualisoiduilla laitteilla sanotaan saavutettavan erittäin hyvä suorituskyky ja tehokkaasti toimiva laitteiston jakaminen eri virtuaalikoneiden kesken [21].

Sen sijaan kernel-kehittäjä ja Kernel-based Virtual Machinen (KVM) ylläpitäjä Avi Kivity kirjoittaa blogissaan [22], että käyttöjärjestelmien muokkaaminen paravirtualisointiin sopivaksi on kuolemassa pois, koska tekniikalla ei saavuteta etua enää nykyään prosessorien tukiessa Nested/Extended Page Tables (NPT/EPT) -muistinosoitusta, joka nopeuttaa muistinkäsittelyä merkittävästi. Itse asiassa paravirtualisoinnissa tehoa kuluu hukkaan, kun virtuaalikoneet keskustelevat VMM:lle suoran laitteisto-osoituksen asemesta.

Paravirtualisointia käyttäviä ohjelmistoratkaisuja ovat mm:

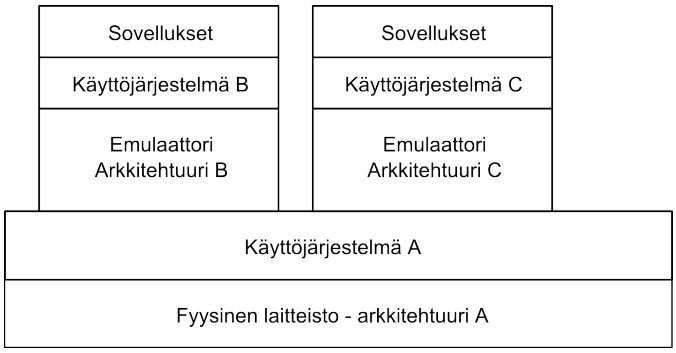
- Xen
- Oracle VM [www-26] (pohjautuu Xen:iin)
- Sun xVM Server [www-27] (pohjautuu Xen:iin)

Uusimmissa toteutuksissa paravirtualisointia käytetään täyden virtualisoinnin yhteydessä asioihin, joissa siitä on selkeästi hyötyä: IO-käsittelyn (etenkin kiintolevyjen) ja oheislaitteiden virtualisointiin. Laitteiston paravirtualisoinnilla vältetään tavallisesti paljon resursseja kuluttava ohjelmistopohjainen laitteiden emulointi, johon liittyy tehottoman binaarimuunnostekniikan käyttäminen (kpl. 3.3.1). Täyden virtualisoinnin ja paravirtualisoinnin yhdistelmää käyttävät mm. VMware ja KVM.

3.3.3 Emulointi

Virtualisoinnin äärimuotona voidaan pitää emulointia, joka tarkoittaa koko laitteistokokonaisuuden simuloimista toisella laitteistoarkkitehtuurilla [19]. Emuloinnin avulla esimerkiksi IA-32-alustalla voidaan ajaa Z80-prosessorille tehtyjä konekielisiä ohjelmia ilman alkuperäisen lähdekoodin uudelleenkääntämistä tai muuttamista. Emulointi sallii resurssien täydellisen hallinnan, mutta hukkaa suuren osan tehosta, koska kaikkien käskyjen pitää kulkea binaarimuunnoksen läpi. Tämän vuoksi tekniikkaa käytetään useimmiten väliaikaisratkaisuihin ja kokeiluihin, esimerkiksi sulautettujen järjestelmien kehitykseen ja -testaukseen PC-koneilla. Kuvassa 9 on esitetty emuloinnin perustoiminta: fyysisen laitteiston päällä voidaan käyttää useaa eri laitteistoarkkitehtuuria, joissa kussakin ajetaan kyseiseen arkkitehtuuriin sopivia käyttöjärjestelmiä ja sovelluksia.

Jatkuvassa tuotantokäytössä olevista emulaattoreista tunnetuin lienee Java Virtual Machine (JVM), joka suorittaa tavukoodin emulointia käytössä olevalle prosessoriarkkitehtuurille. JVM kiertää emuloinnin tehohävikkiä kääntämällä tavukoodin ajonaikaisesti suoraan natiivikoodiksi nk. JIT-kääntäjällä (*Just-In-Time compiler*) [23]. JIT on automaattisesti käytössä Sunin ja BEA:n Java-virtuaalikoneissa, ja ilman sitä Java-ympäristön tehokkuus olisikin monta kertaluokkaa huonompi.



Kuva 9: Emulaattorien toiminta

3.4 Virtuaalipalvelinklusterit

Virtuaalipalvelinklusterilla tarkoitetaan ympäristöä, jossa isäntäpalvelimia (*virtual host*) on useita, ja virtuaalikoneita voidaan siirtää isäntäkoneiden välillä ja hallita keskitetysti. Erityisesti VMware on klustereissa ottanut käyttöön resurssivarasto-käsitteen (*resource pool*), joka tarkoittaa koko klusterin yhteenlaskettuja prosessori- ja muisti- ja massamuistiresursseja. Esimerkiksi käytettäessä isäntäkoneina viittä 2×3.0 GHz palvelinta, joissa kussakin on 16 GiB[6] muistia, saadaan klusterin käytettävissä oleviksi resursseiksi 30 GHz ja 80 GiB RAM. Virtuaalikoneita luotaessa tarvitsee periaattessa vain tarkistaa, että kokonaisresursseja on vapaana – yksittäisistä isäntäkoneista ei tarvitse kantaa huolta.

Merkittävän hyödyllinen ominaisuus koko konesalia virtualisoitaessa on kehittyneemmistä klusteriratkaisuista löytyvä käynnissä olevien virtuaalikoneiden automaattinen siirtäminen isännältä toiselle resurssien käytön mukaan (kuten VMware vMotion Live Migration [24], Sun xVM Ops Center LiveMigration tai RedHat Virtualization Live Migration[7] [www-28]). Tällöin resurssivarasto toimii tarkoituksensa mukaisesti, eli varastossa olevien resurssien määrää voidaan suurentaa tai pienentää suoraan lisäämällä fyysisiä isäntäkoneita, esimerkiksi *blade*- eli korttipalvelimia. Ilman tätä ominaisuutta ongelmia ilmenee, kun virtuaalikoneiden resurssienkäyttö muuttuu: esimerkiksi sovelluspalvelimelle halutaankin antaa kahden gigatavun muistin asemesta kuusi gigatavua, mutta muilta samalla isäntäkoneella ajettavilta virtuaalikoneilta ei pysytytä pienentämään muistia vastaavasti. Automaattisen siirron kanssa järjestelmä siirtäisi joitakin virtuaalikoneita toiselle isännälle niiden toiminnan häiriintymättä. Ominaisuuden hyöty jää kyseenalaiseksi vain, jos virtuaalikoneiden resurssientarve tiedetään tarkalleen etukäteen: tällöin ne voidaan jo luontivaiheessa sijoittaa halutuille isäntäkoneille, eikä niitä jouduta siirtämään jatkossakaan muualle.

Klusterin tehokkaan toiminnan kannalta on erittäin oleellista, että kaikki virtuaalikoneiden levykuvat (*disk image*) ovat levyjärjestelmässä, johon kaikilla isäntäkoneilla on

[6] Ki (2^{10}), Mi (2^{20}) ja Gi (2^{30}) ovat Standardin IEC 60027 mukaisia etuliitteitä
[7] Tukee vain paravirtualisoituja koneita

yhtäaikainen pääsy. Pienehköissä järjestelmissä voitaisiin käyttää paikallisia levyjä ja verkkojakoja, mutta klusteriympäristöissä lähes aina vaaditaan levyiltä suurta tehokkuutta, joten käytetään järeämpiä *Storage Area Network* (SAN) -ratkaisuja. SAN-levyä käytettäessä isäntäkoneet näkevät levyn aivan kuin se olisi paikallinen, vaikka todellisuudessa levyn kanssa keskustellaan esimerkiksi Internet Small Computer Serial Interfacella (iSCSI) Fibre Channelin yli.

SAN-levyjen käyttäminen ei kuitenkaan ole niin suoraviivaista, sillä tavallisia tiedostojärjestelmiä, kuten Third extended file system (ext3) ja NT File System (NTFS), ei ole suunniteltu lohkotasolla usean koneen yhteiskäyttöä varten. Jos tällaista yritetään, tiedostojärjestelmän sekoaminen on lähes varmaa, koska näissä tiedostojärjestelmissä ei ole mitään lohkotason lukitusmekanismeja. Sen sijaan SAN-järjestelmässä käytetään klusteritiedostojärjestelmää (*cluster file system*), kuten IBM General Parallel File System (GPFS) [www-29] tai RedHat Global File System (GFS) [www-30], jotka on suunniteltu usean tietokoneen yhtäaikaisia luku- ja kirjoitusoperaatioita varten. Perinteisesti klusteritiedostojärjestelmät ovat sisältyneet valmistajien kalliisiin suurpalvelinratkaisuihin, mutta kernelistä 2.6.19 lähtien GFS on ollut myös osa Linuxia [25].

Levyjärjestelmien suorituskykyeroja use-case-ympäristössä tutkitaan luvussa 5.7.

3.5 Muistin- ja levynkäyttö

Partitioiduissa järjestelmissä ja alkeellisissa virtuaaliarkkitehtuureissa kullekin itsenäiselle virtuaali- tai osajärjestelmälle piti osoittaa ja varata kiinteä määrä muistia ja levytilaa. Tällöin esimerkiksi isäntäpalvelimella, jossa on 4096 MiB RAM-muistia, voitiin ajaa kolmea virtuaalikonetta, joista kukin käytti 1280 MiB muistia, jättäen pääjärjestelmän käyttöön 256 MiB. Jos jokin näistä virtuaalikoneista tarvitsi oikeasti vain 256 MiB, sillä ei ollut merkitystä, koska muistin varaus oltiin tehty kiinteästi. Sen sijaan edistyneemmillä järjestelmillä vastaavassa tilanteessa muistia voidaan allokoida virtuaalikoneille enemmän kuin sitä oikeasti on käytettävissä (*over commit*) – toivoen, että kaikki virtuaalikoneet eivät kuitenkaan yhtä aikaa käytä maksimimäärää sallitusta muistista [26]. Näin ollen vastaavalle palvelimelle voidaan perustaa esimerkiksi kuusi virtuaalikonetta, joista kullekin asetetaan muistirajaksi sama 1280 MiB.

Levyjen suhteen on myös erilaisia ratkaisuja. Tavallinen toimintamalli uutta virtuaalikonetta perustettaessa on luoda sille kiinteäkokoinen levykuva. Esimerkiksi moderneja käyttöjärjestelmiä varten, ilman suuria sovellusohjelmia, noin kymmenen gigatavun levy on sopiva lähtökohta[8]. Virtuaalikoneohjelmistosta ja isäntäkoneen tiedostojärjestelmästä riippuen varaus voidaan kuitenkin tehdä eri tavoin. Levynopeudeltaan tehokkain,

[8] VMware ESX 3:lla luotavien virtuaalikoneiden oletuslevynkoko on 8 GiB, johon mahtuu hyvin esimerkiksi Windows- tai Linux-perusjärjestelmä

joskin luontivaiheessa hidas tapa on varata saman tien koko 10-gigatavuinen yhtäjaksoinen lohko, mutta tällöin levytilaa kuluu hukkaan aivan kuten RAM-muistin tapauksessa, jos virtuaalikone ei tarvitse levystä kuin osan. Levytilaa tehokkaammin käyttävä ratkaisu on tehdä nk. harva-allokaatio (*sparse allocation*), jossa varattu tiedosto näyttäisi olevan kymmenen gigatavun kokoinen, mutta todellisuudessa käyttää levyä vain niiltä osin kuin tiedostoon on kirjoitettu [27]. Tämän vaarana on kuitenkin suorituskyvyn heikkeneminen pirstaloitumisen (*fragmentation*) myötä, koska tiedostolle ei ole varattu levyltä yhtäjaksoista aluetta. Esimerkiksi Linuxissa ext3-tiedostojärjestelmää käytettäessä kaikki tiedostot ovat lähtökohtaisesti sparse-tyyppisiä. Gigatavun kokoisen, mutta levyltä kuitenkin luontivaiheessa vain 16 kilotavua tarvitsevan tiedoston voi luoda seuraavasti:

```
$ dd if=/dev/zero of=testfile bs=1 count=1 seek=1024M
1+0 records in
1+0 records out
$ find . -name testfile -printf "Koko: %s, koko levyllä: %kk.\n"
Koko: 1073741825, koko levyllä: 16k.
```

Virtuaalikoneista voidaan tehdä tilannekuvia (*snapshot*), joihin tallennetaan virtuaalikoneen levy- ja muistisisältö sellaisenaan. Tämä on hyödyllistä kokeiltaessa esim. jonkin konfiguraatiomuutoksen tai ohjelmistoasennuksen vaikutuksia järjestelmän toimintaan, mahdollistaen helpon ja nopean palautumisen tilaan ennen muutoksia [28]. Tilannekuvia otettaessa käytetään nk. δ-levyjä (delta-levy): tilanteen jäädyttämisen jälkeen alkuperäiseen levykuvaan ei tallenneta mitään, vaan kaikki muutokset tehdään erilliseen δ-levyyn, joka toimii differentiaalisena jatkona alkuperäiselle levylle [29]. Menettely heikentää selvästi suorituskykyä, mutta mahdollistaa tilannekuvan ottamisen ilman aikaa vievää datan kopioimista ja säästää samalla runsaasti levytilaa. Usean tilannekuvan ottamisen jälkeen käytössä on myös yhtä monta δ-levyä, ja suorituskyky voi olla tämän myötä kovin heikko. Jos tilannekuvista luovutaan, saadaan δ-levyt yhdistämällä (*disk consolidation*) jälleen yhtenäinen levykuva.

3.6 Virtuaalikoneiden hallinta

Suurimman käyttäjille näkyvän eron eri virtualisointiratkaisuiden ja -tuotteiden välillä tekee järjestelmäkokonaisuuden hallinta. Yksinkertaisimmillaan on käytettävissä vain komentorivipohjainen työkalu, esim. Xen xm, jolla virtuaalikoneita voi sammuttaa ja käynnistää. Satoihin palvelimiin skaalautuvat ratkaisut sen sijaan tarjoavat erilaisia graafisia hallintavälineitä, kuten VMware ESX Infrastructure Client, Microsoft Virtual Server ja RedHat Virtual Machine Manager tai jopa sovelluskehitykseen räätälöityä Web-pohjaista palvelinryppäiden hallintaa (VMware Stage Manager). Graafisilla välineillä kokonaisuuksien hahmottaminen on helpompaa, vaikka ominaisuuksia ei olisikaan enempää kuin tekstipohjaisessa työkalussa.

Käyttötarkoituksesta riippuen tulee ratkaisua valittaessa kiinnittää huomiota erilaisiin hallinnallisiin ominaisuuksiin. Uusien virtuaalikoneiden perustamisen helppous on oleellista varsinkin testiympäristöissä. Erilaiset pohjat (*template*) ja valmiiden koneiden kloonaus nopeuttavat koneiden perustamista suuresti. Käynnissä olevien virtuaalikoneiden kloonaus (*hot cloning*) helpottaisi työtä entisestään, mutta ominaisuus ei ole suoraan tuettu ainakaan yleisimmissä järjestelmissä (VMware, Microsoft, Sun, Xen). Saman asian saa onneksi toteutettua kiertotoeitse kaikissa tilannekuvia tukevissa virtuaalikoneissa ottamalla koneesta tilannekuvan ja kloonaamalla tämän jälkeen alkuperäiset levykuvat (esim. VMware ESX 3.5:llä [www-31]).

Automaattinen konfigurointi vähentää virtuaalikoneiden luomisen yhteydessä tehtävää manuaalista työtä. Esimerkiksi IP-osoite ja domain-nimi voidaan antaa jo virtuaalikoneen luomiseen käytettävässä työkalussa, joka suorittaa vaadittavat toimenpiteet käyttöjärjestelmätasolla. Samaan toiminnallisuuteen liittyy virtuaalikoneiden hallittu uudelleenkäynnistäminen ja sammuttaminen, jotka voidaan tehdä suoraan hallintatyökalusta. Virtuaalikoneen sisäisen toiminnan ohjaaminen isäntäkoneelta käsin edellyttää aina komentolinkkiä järjestelmien välille. VMware on julkaisemassa suuren osan tarkoitukseen tekemästään WMware Tools -ohjelmistosta vapaaseen kehitykseen projektina "Open Virtual Machine Tools" [www-32]. Sekä VMwarea että muita osapuolia hyödyttäisi suuresti, jos työkalut olisivat saatavilla mahdollisimman monelle eri arkkitehtuurille ja käyttöjärjestelmälle.

Virtuaalialusta tarjoaa yleensä myös jonkinlaisen työkalun virtuaalikoneiden tarkkailuun. Työkalu voi varoittaa esimerkiksi suuresta CPU-käytöstä tai muistin täyttymisestä. Nämä tarkkailuvälineet eivät kuitenkaan korvaa erillistä tarkkailuohjelmistoa kuten Nagiosta [www-33], jolla voidaan esimerkiksi prosessi- tai palvelukohtaisesti seurata palvelimen toimintaa. Matalan tason statistiikan keräämiseltä erikseen kuitenkin voidaan säästyä, jolloin Muninin [www-34] kaltaisia palveluja ei tarvitse asentaa.

Organisaatioympäristöissä merkittävä kriteeri hallinnalle on käyttöoikeuksien hallinta; määrätyt henkilöt halutaan päästää hallitsemaan tiettyä joukkoa virtuaalikoneista tai vain tiettyjä ominaisuuksia. Esimerkiksi klustereissa voi olla yhtä aikaa käynnissä kymmenien eri projektien kehityskäytössä olevia virtuaalipalvelimia, eikä kehittäjien ole sallittua päästä toistensa palvelimille mm. arkaluonteisen tiedon vuoksi. Tuotantokoneille pääsy halutaan myös rajoittaa vain järjestelmänvalvojatason käyttäjiin.

Näiden lisäksi on oleellista, että peruskonfiguraatio-operaatiot, kuten muistin määrän muuttaminen, virtuaalilevyjen lisäys, verkkokortin MAC-osoitteen vaihtaminen ja prosessorien määrän säätäminen ovat vaivattomia ja nopeita, ja virheiden tekemisen riski on mahdollisimman pieni.

3.7 Tunnettuja ongelmia

Organisaatioissa voi olla käytössä moniin eri prosessoriarkkitehtuureihin pohjautuvia järjestelmiä, joita ei yleensä voida asentaa samaan virtuaaliklusteriin. Joissain tapauksissa eri käyttöjärjestelmillä, käyttöjärjestelmäversioilla ja jopa sovelluksilla on yhteensopivuusongelmia tiettyjen virtuaalialustojen kanssa, tai valmistajat suostuvat tukemaan vain valitsemiaan kombinaatioita. Esimerkiksi Oracle ei tue järjestelmiensä ajamista VMwarella:

> "Oracle has not certified any of its products on VMware virtualized environments. Oracle Support will assist customers running Oracle products on VMware in the following manner: Oracle will only provide support for issues that either are known to occur on the native OS, or can be demonstrated not to be as a result of running on VMware." [30]

Tukiongelma on itse asiassa hyvin merkittävä, koska juuri koskaan ei haluttaisi ajaa monia eri virtualisointialustoja, ja klusterissa tämä on luonnollisesti mahdotontakin. Näin ollen valittu alusta rajaa suoraan tietyt ohjelmistot pois ja aiheuttaa organisaation urautumista tietyn toimittajan tuotteisiin. Tällä voi taas olla suuria kustannusvaikutuksia sekä ohjelmistolisenssien että henkilöstön koulutuksen muodossa.

Tietyn minimipalvelutason (*minimum service level*) ylläpitäminen virtuaaliympäristössä on usein vaikeaa, koska järjestelmät eivät takaa millekään virtuaalikoneelle tiettyä määrää dynaamisesti jaettavista resursseista, kuten kellojaksoista, IO-väylästä tai verkkokapasiteetista. Fyysisiä prosessoriytimiä ollessa vähemmän kuin ajettavia virtuaalikoneita prosessoritehoa ei raskaassa kuormitustilanteessa yksinkertaisesti riitä jokaiselle virtuaalikoneelle tehokkaan toiminnan edellyttämää määrää. Tämän takia virtualisointi ei sovellu tilanteisiin, joissa ei sallita viiveitä sekä ajoittain heikkoa suorituskykyä. Tällaisia ovat mm. erilaiset ohjausjärjestelmät sekä reaaliaikasovellukset.

Virtuaaliympäristöön siirryttäessä täytyy kartoittaa virtualisoinnin aiheuttamat riskit. Koska samalla fyysisellä palvelimella ajetaan jopa kymmeniä virtuaalipalvelimia, aiheuttaa fyysisen koneen vikaantuminen kaikkiin näihin palveluihin katkoksen. Hallinnon päätettäväksi jää, kuinka suuri riski ollaan valmis ottamaan kustannussäästöjä haettaessa, tai miten paljon redundanssia ollaan valmiita rakentamaan palveluiden turvaamiseksi.

Virtualisointikerros aiheuttaa myös tietoturvariskejä: sekä Core Security Technologies että IntelGuardians ovat jo löytäneet VMwaren tuotteista haavoittuvuuksia, joiden avulla virtuaalikoneen sisältä käsin on mahdollista hallita osittain muidenkin virtuaalikoneiden toimintaa [31] [10]. Mainitut yritykset ohjeistavatkin organisaatioita konfiguroimaan verkkonsa niin, että virtuaalikoneiden haavoittuvuuksia ei voida hyödyntää tietomurtoihin.

Monissa organisaatioissa ohjelmistolisenssien hallinta on havaittu vaikeaksi virtuaaliympäristössä, jossa koneista tehdään kopioita ja uusia koneita perustetaan ja vanhoja tuhotaan tiheään. Ohjelmistoyritykset eivät yksinkertaista tilannetta tarjoamalla virtuaaliympäristöihin erilaisia lisenssejä ja lisensointimalleja, vaikka näillä yleensä kustannuksia säästetäänkin: esimerkiksi RedHat-virtualisointia käytettäessä voidaan yhdellä virtualisointilisenssillä perustaa kuinka monta RedHat Enterprise Linuxia (RHEL) käyttävää virtuaalikonetta tahansa, mutta VMwarea käytettäessä pitää hankkia RHEL for VMware -lisenssejä, jotka ovat huomattavasti kalliimpia [Pekkarinen, RedHat Finland].

3.8 Sopivan tuotteen valinta

Organisaation palvelinratkaisua valittaessa on pohdittava kaikkia edellä esitettyjä asioita. Ensin ratkaistaan yleiset kysymykset, kuten "millaisia palveluita ja järjestelmiä organisaatiossa on", "millaisia palveluita tulevaisuudessa tarvitaan", "kuinka paljon on talon sisäisiä ja mahdollisesti ulkopuolisia käyttäjiä" sekä "miten nykyiset järjestelmät integroidaan uuteen". Vasta tämän jälkeen voidaan käydä läpi kapasiteettiin, tehokkuuteen, skaalautuvuuteen ja tuotetukeen liittyvät tekniset asiat sekä suhteuttaa ne käytettävissä olevaan budjettiin.

Tuotteen valinta ilman kunnollista taustoihin perehtymistä voi johtaa hankaliin ongelmiin, kuten minulla kerran kävi:

> Suunnitellessani Tietoverkkotekniikan laboratoriotyöt -kurssille uutta laboratoriotyötä Virtual Private Networks (VPN) -aiheesta Q4/2007–Q2/2008 otin fyysisten koneiden määrän vähentämiseksi käyttöön virtuaalikoneet. Ilman tätä työhön olisi tarvittu viisi konetta, joille laboratoriossa ei ollut tilaa. Ubuntu tuki vielä syksyllä 2007 Xen-paravirtualisointia, joten ajattelin sen olevan helppo ja yksinkertainen vaihtoehto, koska isäntäkoneella oli jo toimiva Ubuntu-järjestelmä.
>
> Ubuntun paketoimassa Xen-järjestelmässä ilmeni kuitenkin hyvin pian vakavia ongelmia käynnistysskriptien syntaksivirheistä lähtien aina kriittisiin kernelin virheisiin. Osaan löytyi korjauksia Ubuntun virallisilta webbifoorumeilta, ja osaa, kuten kernelin korjausta virtuaaliverkkorajapintavirheelle, sai etsiä mm. japanilaisen Ubuntu-yhteisön sivuilta. Osan virheistä jouduin korjaamaan itse.
>
> Virhekorjauksia etsiessäni törmäsin mm. usean aktiivikäyttäjän havaitsemaan Ubuntun Xen-kernelin aikaongelmaan, joka jumiutti systeemin täysin, mutta johon kukaan ei ollut osannut tehdä korjausta vielä puoli vuotta ongelman raportoimisen jälkeenkään. Osa oli ratkaissut ongelman käyttämällä omaa tai RedHat Linuxin kerneliä, osa oli vaihtanut toiseen käyttöjärjestelmään tai virtuaalialustaan.

Ongelmallista minun kannaltani tilanteessa oli, että Ubuntun tai Xenin vaihtaminen toiseen systeemiin olisi vaatinut koko järjestelmän rakentamisen alusta alkaen uudestaan, mihin en halunnut aikataulusyistä ryhtyä. Onneksi sain kokonaisuuden toimimaan tyydyttävästi pahimmat virheet korjattuani. Ubuntu lopetti Xenin tukemisen kesällä 2008.

Taulukosta 1 käy ilmi yleisimpien x86-alustalla toimivien virtualisointituotteiden ominaisuuksia. Taulukon perusteella ei voida tehdä organisaation virtualisointituotevalintaa, mutta siitä käy ilmi, miten erilaisia ratkaisuja on tarjolla sekä avoimen että suljetun lähdekoodin kehittäjiltä.

Tuote	Kehittäjä	Lisenssi	Tuki	Tekniikka	CPU-tuki	Isäntäkoneen käyttöjärjestelmä	Tuetut asiakas-käyttöjärjestelmät
Hyper-V	Microsoft	Kaupallinen, sisältyy käyttöjärjestelmään	Lisämaksusta	Täysi virtualisointi	x86-64	Windows Server 2008	Windows 2000, 2003, 2008, XP, Vista, Linux (SUSE 10)
KVM	Qumranet, Inc	GPL v2	Ei saatavilla	Täysi virtualisointi + laitteiden paravirtualisointi	x86-32, x86-64, IA64, s390, PowerPC	Linux	-
Virtual Server 2005	Microsoft	Ei vapaa, mutta tuote ilmainen	Lisämaksusta	Täysi virtualisointi	x86-32, x86-64	Windows Server 2003, Windows XP	Windows NT, 2000, 2003, Linux (RedHat, SUSE)
QEMU	Fabrice Bellard & co	GPL / LGPL	Ei saatavilla	DRC [1]	x86-32, x86-64, IA64, s390, PowerPC, Alpha, Sparc, ARM, M68k	Windows, Linux, Mac OS X, Solaris, FreeBSD, OpenBSD, BeOS	-
RedHat Virtualization	RedHat (perustuu Xeniin)	Kaupallinen, sisältyy käyttöjärjestelmään	Sisältyy hintaan	Paravirtualisointi	x86-32, x86-64	RedHat Enterprise Linux	RedHat Enterprise Linux
Sun xVM server	Sun Microsystems	GPL v2, enterprise-versio maksullinen	Lisämaksusta	Täysi virtualisointi + paravirtualisointi	x86-32, x86-64	Itsenäinen järjestelmä	Soalris, OpenSolaris, RHEL, Windows 2003, Windows 2008
VMware ESX 3.5	VMware	Kaupallinen, $0–$5750	Lisämaksusta	Täysi virtualisointi + laitteiden paravirtualisointi	x86-64	Itsenäinen järjestelmä	Windows, Linux, Netware, Solaris, FreeBSD
VMware Server 2.0	VMware	Ei vapaa, mutta tuote ilmainen	Lisämaksusta	Täysi virtualisointi	x86-32, x86-64	Linux, Windows	DOS, Windows, Linux, FreeBSD, Netware, Solaris
Xen	Citrix Systems	GPL	Lisämaksusta	Paravirtualisointi	x86-32, x86-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP, 2003

[1] Dynamic Recompilation, osittainen ajonaikainen uudelleenkääntäminen

Taulukko 1: Yleisimpien x86-virtualisointituotteiden ominaisuuksia

3.9 Yhteenveto

Lähes kaikki merkittävät käyttöjärjestelmävalmistajat sekä myös suuri määrä muita ohjelmistoyrityksiä ovat tuoneet markkinoille kirjavan joukon virtuaalialustoja. Tekniikat poikkeavat toisistaan joko erittäin merkittävästi (vrt. Xen ja VMware ESX) tai ovat käytännössä samoja (vrt. Xen ja RedHat Virtualization), mutta eri nimillä markkinoituja. Viimeaikoina on ollut havaittavissa jonkin verran yritystä yhdistää eri tekniikoiden parhaita puolia, kun esimerkiksi Xen mahdollistaa täyttä virtualisointia käyttäen minkä tahansa käyttöjärjestelmän ajamisen, ja toisaalta VMware on ottanut käyttöön paravirtualisointiominaisuuksia laitteiston käsittelyä nopeuttaakseen.

Suuria palvelinsaleja virtualisoitaessa täytyy ottaa huomioon paljon enemmän asioita kuin yksinkertaisessa työpöytäkäytössä. Merkittäviksi kysymyksiksi nousevat hallittavuus, toimintavarmuus, ohjelmistojen tuki, itse järjestelmän käyttötuki, kokonaiskustannukset, valvonta, tietoturva sekä klusterointiominaisuudet. Alustan vaihtaminen toiseen jälkikäteen on todella raskas prosessi, joten valinta pitää tehdä pitkälle tulevaisuuteen katsoen.

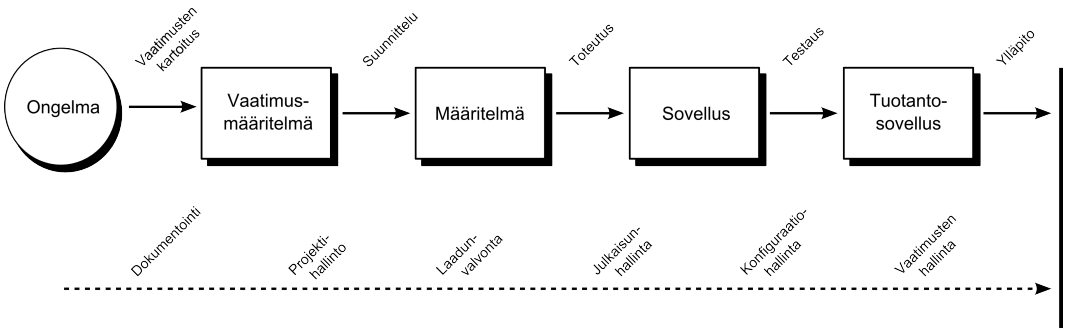
4 Sovelluskehitysympäristön virtualisointi

Sovelluskehitysympäristö kattaa kaikki tietojärjestelmät, joita tarvitaan sovelluksen kehityskaaren aikana. Tässä luvussa keskitytään sovelluskehitysprosessin loppupään vaiheisiin – testaukseen, käyttöönottoon ja ylläpitoon – joissa palvelinalustalla on suurin merkitys sovelluksen toimintaan ja kehitysprosessin hallintaan.

Luvussa käydään läpi sovelluskehitysprosessin sekä -ympäristön suunnittelun vaiheita, vertaillaan perinteisen palvelinympäristön ja virtuaaliympäristön eroavaisuuksia sekä esitetään tapoja, joilla asioita voi tehdä tehokkaammin virtualisoinnin avulla.

4.1 Sovelluskehitysprosessi

Sovelluskehitysprojektien läpivientiin on kehitetty lukuisia erilaisia malleja, ja lisäksi projektit vaativat malleihin omat kustomointinsa. Kehitysprosessin tarkasta kulusta riippumatta sovellusten kehitykseen liittyvät lähes aina kuvassa 10 esitetyt vaiheet. Todellisuudessa vaiheet eivät ole näin tarkasti rajattuja ja sama vaihe voi toistua useasti. Tässä luvussa noudatetaan van Vlietin kuvausta yleisestä kehitysprosessista [32].



Kuva 10: Sovelluskehitysprojektin elinkaari yksinkertaistettuna. Yllä sovelluksen kehitysvaiheet sekä alla koko projektin kestoiset toiminnot.

Kehitysvaiheet

Ensimmäisessä vaiheessa määritetään ja dokumentoidaan sovellukseen sekä sen käyttöön ja kehitykseen liittyvät vaatimukset. Vaatimukset voidaan jakaa kolmeen kategoriaan: toiminnallisiin vaatimuksiin, ei-toiminnallisiin vaatimuksiin (esim. sovelluksen suorituskykyyn, luotettavuuteen ja käytettävyyteen liittyvät parametrit) sekä rajoitteisiin (esim. standardit, organisaation politiikat ja laitteistorajoitteet). Prosessin tuloksena kirjoitettava vaatimusmäärittely (*requirements specification*) toimii pohjana kaikille seuraaville kehitysvaiheille.

Suunnitteluvaiheessa luodaan määrittelyn pohjalta sovelluksen arkkitehtuurinen määritelmä. Sovellus jaetaan komponentteihin ja kuvataan rajapinnat, joilla komponentit

keskustelevat keskenään. Suunnitteluvaiheessa ei tulisi kiinnittää huomiota yksittäisten komponenttien toteutukseen, vaan muodostaa sovelluksen kokonaiskuva ja valita sopivat pohjaratkaisut, jotta kokonaisuus on toteutuskelpoinen. Lopputuloksena saadaan määritelmädokumentti (*specification*).

Toteutuksessa keskitytään yksittäisiin komponentteihin. Ennen ohjelmointia jokainen komponentti täytyy kuitenkin suunnitella: ikään kuin suunnitteluvaihe suoritettaisiin uudelleen komponenttitasolla. Komponenttien ohjelmoinnin jälkeen ne integroidaan toimivaksi kokonaisuudeksi, minkä tuloksena saadaan ajettava ohjelma sekä dokumentoitu lähdekoodi.

Testausta ei varsinaisesti voi erottaa omaksi vaiheekseen, koska niin suuri osa siitä tehdään toteutuksen aikana. Testauksen V-mallin mukaan jokaiselle kehitysprosessin vaiheelle on myös oma testausvaihe: kompontit testataan yksikkötesteissä, arkkitehtuuri integraatiotestissä, toiminnallinen määritelmä järjestelmätestauksessa ja vaatimusmäärittelyn toteutuminen hyväksymistestauksessa [7].

Sovelluksen käyttöönoton jälkeen tapahtuvassa ylläpidossa tarkkaillaan sovelluksen toimintaa ja korjataan mahdollisesti löytyvät virheet. Mahdollisesti nousee esiin asioita, joita pitää lisätä vaatimuksiin tai muuttaa suunnitelmassa. Näiden seurauksena palataan uudelleen kartoittamaan vaatimuksia, suunnittelemaan, toteuttamaan ja testaamaan.

Koko projektin kestoiset toiminnot

Projektin hallinnollisiin tehtäviin kuuluvat projektin läpiviennin suunnittelu, tiimien organisointi, laadunvarmistuksesta huolehtiminen, aikataulun arviointi ja sen toteutuminen sekä jossain määrin rahoitukselliset ja hinnoittelukysymykset. Mitä suurempi ja monimutkaisempi toteutettava sovellus on, sitä enemmän projektipäälliköltä vaaditaan hallinnollista osaamista. Modernit, ketterät prosessimallit edellyttävät myös sekä projektin vetäjältä että sovelluskehittäjiltä paljon enemmän kuin esimerkiksi perinteinen vesiputousmalli [33].

Yksi oleellisimmista koko projektin elinkaaren kestoisista toiminnoista on dokumentointi. Käytännössä kaikkiin ohjelmistoprojekteihin kuuluu vaatimusmäärittelmä, määrittelydokumentti, arkkitehtuurin kuvaus, komponenttien dokumentointi sekä testaussuunnitelma. Dokumentoinnin kattavuus kuitenkin vaihtelee suuresti sekä sovelluksen luonteesta että käytettävästä kehitysprosessista riippuen. Kriittisille, vesiputousmallin mukaan toteutettaville sovelluksille luodaan erittäin tarkka dokumentaatio jokaisesta yksittäisestä komponentistakin, kun taas ketterillä malleilla läpiviedyn peliprojektin moduulien dokumentaatio voi rajoittua lähinnä lähdekoodikommentteihin.

Laadunvalvonnan (*quality assurance*) tarkoituksena on systemaattisesti suoritettavilla laadunvarmistusmenetelmillä antaa riittävän hyvä varmuus, että sovellus täyttää sen vaatimuksissa esitetyt kriteerit[9]. Laadunvarmistusmenetelmät pyrkivät ennaltaehkäisevästi varmistamaan, että sovittuja kehitysprosesseja ja -käytäntöjä noudatetaan. Laadunvalvonta tähtää aina *riittävän hyvään laatuun*, jonka määritelmä vaihtelee mm. sovelluksen kriittisyyden mukaan.

Vaatimusten, julkaisujen, konfiguraatioiden, muutosten ja versioiden hallinta liittyvät muuttuviin vaatimuksiin: millä tavalla menetellään, jos sovelluksen halutaankin toimivan eri tavalla kuin alkuperäisissä vaatimuksissa määriteltiin? Vaatimukset voivat muuttua jo ennen kuin sovelluksesta on tehty yhtään toimivaa versiota tai vasta sovelluksen oltua jo pitkään tuotantokäytössä. Käytettävä sovelluskehitysmalli sanelee suurelta osin lähtökohdat: jos asiakas pyytää lisäämään tuotantosovellukseen toiminnon, kuinka suuri osa sovelluskehitysprosessista pitää käydä uudestaan läpi, että muutos saadaan toteutettua ja julkaistua uudessa tuotantoversiossa?

4.1.1 Vesiputousmalli

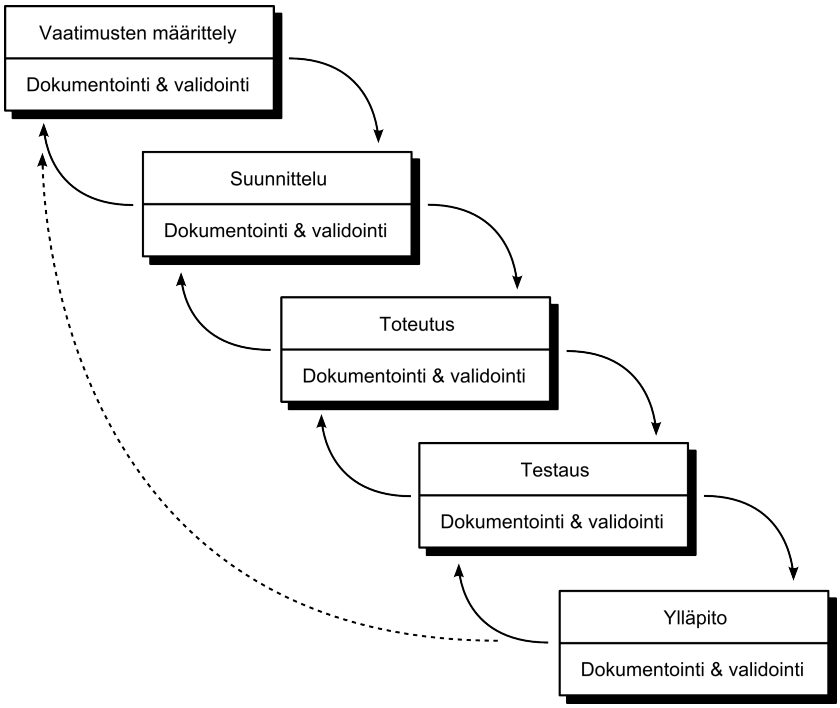
Etenkin monimutkaisia ja toimintakriittisiä sovelluksia kehitetään edelleen vesiputousmallin mukaan. Mallia käytettäessä sovelluskehitysprosessi on jaettu tarkasti edellä lueteltuihin vaiheisiin, joista jokainen dokumentoidaan kattavasti ja hyväksytään ennen seuraavaan siirtymistä (kuva 11). Tällä tavoin prosessin toivotaan pysyvän hallinnassa niin, ettei lopulliseen sovellukseen pääse ominaisuuksia, joita ei ole testattu ja varmistettu sovittujen käytäntöjen mukaisesti. Malli voisi toimia hienosti, jos vaatimukset ovat tarkalleen tiedossa etukäteen, mutta vaatimusten muuttuessa joudutaan palaamaan ensimmäiseen vaiheeseen ja kulkemaan uudestaan kaikkien vaiheiden läpi, ennen kuin ohjelmasta saadaan ulos uusi versio.

Käytännössä on kuitenkin havaittu, että vesiputousmalli ei toimi lähellekään niin ideaalisesti kuin paperilla. Zelkowitzin tutkimuksessa esimerkiksi alle puolet suunnittelusta tehtiin oikeasti suunnitteluvaiheessa. Noin kolmasosa suunnittelusta siirtyi tehtäväksi ohjelmoinnin lomassa ja yli 15 % vielä integraatiotestauksessa ja sen jälkeen, kun sovelluksen olisi pitänyt olla jo valmis [32].

4.1.2 Ketterät mallit

Vesiputousmalli, kuten muutkin raskaat mallit, perustuvat huolelliseen suunnitteluun ja dokumentointiin. Kerran suunniteltua ja päätettyä asiaa on työlästä muuttaa, koska se edellyttää käytännössä kaikkien siihen perustuvien suunnitelmien ja dokumenttienkin

[9] Määritelmä: IEEE Standard Glossary of Software Engineering Terminology



Kuva 11: Vesiputousmalli. Jokainen vaihde dokumentoidaan ja hyväksytään erikseen.

muuttamista. Käytännössä projektit kuitenkin kokevat jatkuvasti muuttuvia vaatimuksia ja tavoitteita, minkä johdosta raskailla malleilla projektiin alunperin suunniteltu aikataulu, hinta ja elinkaari eivät pidä paikkaansa. Näiden ongelmien voittamiseksi on kehitetty nk. ketterät mallit (*agile models*) [32].

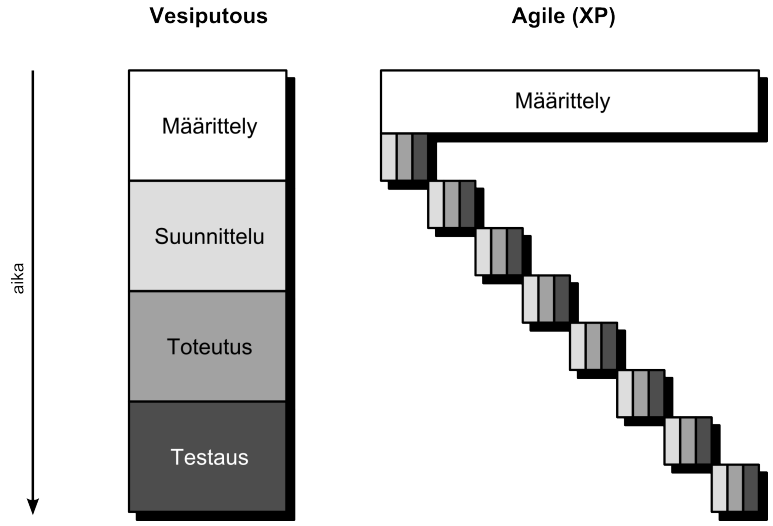
Manifesto for Agile Software Development [www-35] määrittelee ketterien mallien kivijalan seuraavasti:

- **Yksilöt ja vuorovaikutus** tärkeämpiä kuin prosessit ja työkalut
- **Toimiva ohjelma** tärkeämpi kuin kattava dokumentaatio
- **Asiakasyhteistyö** tärkeämpää kuin tarkan sopimuksen neuvotteleminen
- **Muutokseen vastaaminen** tärkeämpää kuin suunnitelman noudattaminen

Perusajatus siis on, että resursseja ei kannata kuluttaa dokumentaation, suunnitelmien ja sopimusten tekemiseen, koska niihin lähes varmasti tulee muutoksia, jotka edellyttäisi niiden uudelleenkirjoittamista useaan kertaan [32]. Ennemmin pyritään saamaan nopeasti valmiiksi toimiva ohjelma. Vaikka se ei kaikkia vaatimuksia täytäkään, tarjoaa se asiakkaalle kuitenkin paljon enemmän arvoa kuin pelkkä suunnitelma tai dokumentti. Ennen kaikkea toimivasta ohjelmasta nähdään helposti, onko projekti kulkemassa oikeaan suuntaan, ja mitä pitää lisätä tai muuttaa seuraavaksi, jotta päästään lähemmäs lopullista tavoitetta.

Ketteriä malleja käyttämällä päästään siis lähemmäs asiakkaiden tarpeita vastaavaa ohjelmistoa siitäkin huolimatta, että tarpeet muuttuvat kesken projektin. Samalla inkrementteinä tapahtuva kehitys pienentää projektin epäonnistumisen riskiä, nopeuttaa kehitystä ja vähentää kustannuksia. Projektihallinnon kannalta tilanne on kaksijakoinen: kokonaisuus on helpompi hallita pieninä paloina, mutta toisaalta alkumäärittely tehdään käytännössä aina koko ohjelmistoa koskien, jolloin usean "aliprojektin" hallinta monimutkaistaa projektipäällikön tehtäviä. Suuren, yli 50 henkilön ohjelmistoprojektin hallinta voi olla vaikeaa ja epätehokasta ketterillä malleilla [32].

Vesiputousmallin ja ketterien mallien konkreettinen ero näkyy kuvassa 12. Esimerkkinä käytetään Extreme programming (XP) -mallia, joka on eräs puhtaimmista agilemalleista. Vesiputouksessa kukin prosessin vaihe käydään läpi vain kerran, kun taas XP:ssä alkumäärittelyn jälkeen kehitys tapahtuu pieninä toistuvina jaksoina, joissa jokaisessa tehdään suunnittelua, toteutusta ja testausta.



Kuva 12: Vesiputousmallin ja ketterien mallien ero projektin elinkaaren suhteen

4.2 Sovelluksen julkaisunhallinta

Sovelluksen julkaisunhallinta (*Software Release Management*, SRM) kattaa sovelluksen eri versioiden julkaisun kehitys-, testaus- ja tuotantovaiheiden läpi sovelluksen koko elinkaaren ajan. Julkaisunhallintaan kuuluu myös päätökset mm. julkaisut tekevistä henkilöistä, julkaisuformaateista ja -medioista ja versioinnista [34]. Tämän työn kannalta on oleellista, millä tavalla virtualisoitua ympäristöä voidaan hyödyntää SRM-prosesseihin, helpottaen erityisesti sovelluksen käsittelyä kehitys- ja tuotantoversioiden välillä.

Julkaisunhallinnan ja virtualisoinnin yhdistämisestä ei ole saatavilla käyttökelpoista tutkimusmateriaalia. Näin ollen tämän luvun tiedot perustuvat suurimmaksi osaksi J2EE-sovelluskehittäjien sekä virtuaaliympäristökonsultin haastatteluihin. Lisäksi VMware

on huomioinut julkaisunhallinnan Stage Manager-tuotteessaan [www-36], joka helpottaa kehityskaaren eri vaiheissa eli tasoilla (*stage*) olevien palvelimien hallintaa.

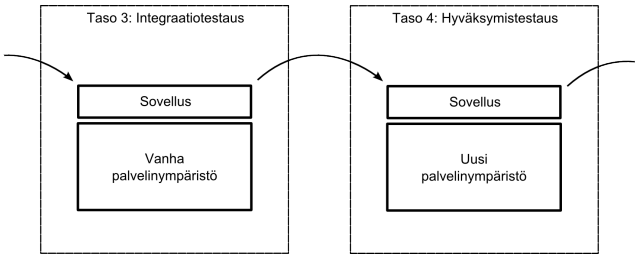
Sovelluskehitysympäristön rakentaminen niin, että se on sekä sovelluskehittäjien prosesseja noudattava (luku 2.3, kuva 3) että palvelimien ylläpitäjien kannalta järkevästi hallittavissa, on sovelluskehitysympäristön virtualisoinnin suurimpia haasteita. Suunnittelussa pyritään toteuttamaan seuraavat lähtökohdat:

- Sovelluksen siirtäminen prosessissa seuraavalle tasolle on helppoa
- Ympäristöillä on selvärajainen käyttötarkoitus (esim. hyväksymistestaus ennen tuotantoonottoa)
- Uuden ympäristön perustaminen on nopeaa
- Prosessi noudattaa selkeää kiertokulkua
- Itse prosessista ei aiheudu sovelluskehittäjille eikä palvelimien ylläpitäjille kohtuutonta työtaakkaa

Fyysisiä palvelimia käytettäessä sovelluksen kuljettaminen tasolta toiselle (kuva 13) sovelluskehitysprosessin mukaisessa kiertokulussa on haastavaa. Sovelluksen julkaiseminen seuraavaan ympäristöön, esimerkiksi siirtäminen integraatiotestauksesta hyväksymistestaukseen, sisältää useita toimenpiteitä, joista moni tehdään usein käsin. Vaadittuja työvaiheita ovat:

1. Ympäristöasetusten, kuten tietokantaosoitteiden ja käyttäjätunnistustavan muuttaminen uutta ympäristöä vastaaviksi
2. Sovelluksen kääntäminen muutetuilla asetuksilla
3. Uuden ympäristön tyhjentäminen edellisistä sovelluksista; mahdollisesti koko järjestelmän uudelleenasennus, mikäli testauksessa on kertynyt paljon väliaikaismateriaalia
4. Taustajärjestelmien, kuten tietokannan siirtäminen uuteen ympäristöön
5. Uuden palvelimen palvelinasetusten, kuten käyttöjärjestelmäpäivitysten, muuttaminen edellistä tasoa vastaaviksi
6. Sovelluksen asentaminen uuteen ympäristöön

Lisää manuaalista työtä ja jopa ongelmia seuraa, jos käytettävän sovelluskehitysmallin mukaan voidaan tehdä siirtoja takaisin tuotannosta testiin, testistä kehitykseen tai näiden sisällä ylemmältä tasolta alemmalle. Tällöin edellisellä tasolla voi olla jo asennettuna sovelluksen uudempi versio, joten pitäisikö sen testaus lopettaa ja korvata ympäristö ylemmältä tasolta palautettavalla ympäristöllä?



Kuva 13: Sovelluksen siirtäminen testauksessa edelliseltä tasolta seuraavalle

Virtualisointi mahdollistaa palvelinkokonaisuuksien tai jopa palvelinrypäskokonaisuuksien siirtämisen tasolta toiselle sellaisenaan [35]. Näennäisesti voidaan eliminoida kaikki edellä listatut manuaaliset työvaiheet, jos sovelluksen siirtämisen sijaan siirrettäisiinkin kokonainen palvelin – esimerkiksi vain vaihtamalla palvelimen verkko-osoite seuraavan tason käyttämään aliverkkoon. Palvelimen palauttaminen edelliselle tasolle ei myöskään ole ongelma, koska samalla tasolla voi aivan hyvin toimia useita palvelimia yhtäaikaisesti. Toimivaksi todettu tuotantopalvelin voidaan kopioida lähtökohdaksi uudelle kehitysversiolle käden käänteessä [35].

Vanhassa järjestelmässä on lisäksi vaarana, että palvelimien konfiguraatiot muuttuvat hiljalleen epäyhtenäisiksi, jonka jälkeen sovelluksen toimiminen yhdessä ympäristössä ei enää takaa sen toimimista toisessa. Tätä varten kaikki ohjelmistoasennukset, -päivitykset, palvelinohjelmistojen konfiguraatiomuutokset sekä tietokantakonversiot on tehtävä erittäin tarkasti määrätyn kaavan mukaan jokaiselle ympäristölle ja pidettävä tarkkaa dokumentaatiota kuhunkin palvelinkoneeseen tehdyistä asennuksista. Tästäkin huolimatta käytäntö on osoittanut, että viimeistään vuoden–kahden kuluttua konfiguraatiot ovat muuttuneet oleellisesti erilaisiksi. Virtuaalisia palvelinkokonaisuuksia käsiteltäessä näitä ongelmia ei ole.

Sovelluskehittäjien haastattelussa kävi kuitenkin ilmi, että myös uuteen prosessiin liittyy vakavia ongelmia. Suurimpana ongelmana kehittäjät pitävät sitä, että vaikka palvelimien siirto poistaa joitakin manuaalisia työvaiheita, se itse asiassa luo enemmän lisää uusia. Etenkin palvelimen siirto hyväksymistestistä tuotantoon on vaikea: tuotantoa varten palvelimelle pitää asettaa tiukat tietoturva-asetukset, poistaa kaikki testiohjelmistot, kääntää sovelluksesta tuotantoversio, käydä läpi palvelimen käyttäjätilit sekä muut testausta varten luodut asetukset, vaihtaa testitaustajärjestelmien viittaukset (mm. tunnistusjärjestelmä, tietokanta ja postipalvelin) tuotantojärjestelmiin ja asentaa monitorointi- ja varmuuskopioagentit. Kun sovellus palautetaan takaisin tuotannosta testiin, tai kun tuotantosovelluksesta otetaan kopio kehityspohjaksi uudelle versiolle, pitää kaikki edellä mainitut muutokset jälleen korvata testiasetuksilla.

Käytännössä tehokkain ratkaisu on yhdistää vanhaan malliin joitakin virtualisoinnin tuomia etuja. Luvussa 5.5 esitetään yksi tällainen toimintamalli.

4.3 Sovelluskehityksen ulkoistaminen virtualisoinnilla

Organisaation ulkopuolisia sovellustoimittajia käyttämällä halutaan siirtää kehitystä pois organisaatiosta, jotta voidaan keskittyä paremmin omiin ydinliiketoiminta-alueisiin. Sovelluskehityksen ulkoistaminen tarkoittaa myös fyysistä kehitysympäristön ja mahdollisimman pitkälti myös testiympäristön siirtämistä pois organisaation tiloista ja palvelimilta.

Sovelluskehittäjät tarvitsevat kuitenkin kehitykseen ja etenkin testaukseen tuotantoympäristön kaltaisen järjestelmän tarjoamaan sovellukselle taustapalvelut sekä oikeanlaista dataa. Jos kehittäjien ympäristö poikkeaa merkittävästi organisaation oikeasta ympäristöstä, testitulosten ei voida olettaa olevan päteviä ympäristöjen välillä. Virtualisointi tarjoaa ratkaisuksi mahdollisuutta kopioida koko kehitysympäristö sellaisenaan sovellustoimittajalle.

Virtuaalikoneiden siirtäminen on periaatteessa helppoa: kopioidaan koneiden konfiguraatiotiedostot sekä virtuaalilevyt. Käynnissä olevan koneen siirtämiseen voi soveltaa luvussa 4.6 esitettyä tilannekuviin perustuvaa tekniikkaa. Ongelma kuitenkin on virtualisointiarkkitehtuurien epäyhteensopivuus. Kaikki valmistajat, ja jopa saman valmistajan eri tuotteet, ovat käyttäneet erilaisia konfiguraatioita ja virtuaalilevyjä. Näin ollen sovellustoimittajalla pitäisi olla täsmälleen samanlainen alusta kuin itse organisaatiolla. Asiaa helpottamaan on kehitty avointa standardia Open Virtualization Format (OVF).

4.3.1 Open Virtualization Format

Virtualisoinnin markkinajohtajien (Xen Source, VMware, IBM, HP, Dell ja Microsoft) viimeaikaisen yhteistyön tuloksena on saatu aikaan DMTF-standardi[10] avoimesta formaatista virtuaalikoneiden paketointiin ja jakeluun [36]. OVF ei täysin korvaa valmistajakohtaisia ratkaisuja, vaan pikemminkin käärii nämä standardinmukaiseen XML-pohjaiseen formaattiin, joka voidaan purkaa millä tahansa standardia tukevalla virtuaalialustalla takaisin alustan omaan formaattiin. Toki alusta voi tukea myös virtuaalikoneiden käyttämistä suoraan OVF-paketeista.

Open Virtualization Formatin tarkoitus on mahdollistaa virtuaalikone- ja sovelluskokonaisuuksien (*virtual appliance*) siirtämisen joustavasti ympäristöstä toiseen, esimerkiksi sovelluskehittäjiltä asiakkaalle. Tällä tavoin voidaan luoda testattuja ja sertifioituja kokonaisuuksia, jolloin asiakkaan eri tarvitse sovellusta ostaessaan ja käyttöön ottaessaan huolehtia lainkaan palvelinympäristöstä; aivan kuin ostaisi valmiiksi kalustetun talon.

[10] Distributed Management Task Force, Inc. [www-37]

OVF-formaatti on suunniteltu laajennettavaksi sitä mukaa, kun virtuaalitekniikka kehittyy ja ominaisuudet lisääntyvät. Tuettuna ovat jo nyt sekä yhden että usean virtuaalikoneen konfiguraatiot, lokalisointi, pakettien validointi, monet virtuaalilevyformaatit sekä yksinkertainen lisensointiskeema. Lisäksi formaatti sallii laitteisto- ja valmistajakohtaiset erityiskonfiguraatiot [36].

OVF-paketoitujen virtuaalikoneiden siirrettävyys alustalta toiseen voi kuitenkin kohdata haasteita, koska standardi ei aseta alustoille juuri mitään konkreettisia vaatimuksia. Lisäksi OVF-standardia kehitetään erillään virtuaalialustoista, minkä myötä uudet ominaisuudet otetaan alustoihin ja standardiin epäsynkronoidusti. Ongelmia voi syntyä esimerkiksi epäyhteensopivista virtuaalilevyistä, prosessoriarkkitehtuurista ja virtualisoidusta laitteistosta.

Siirrettävyydelle on pyritty luomaan pohjaa määrittelemällä standardissa kolme eri siirrettävyystasoa. Siirrettävyystasoa yksi noudattavat paketit toimivat vain täsmälleen tietyllä kokoonpanolla, tason kaksi paketteja voidaan siirtää tuoteperheen sisällä (esim. VMware ESX $\leftrightarrow$ VMware Server) ja tasolla kolme vaaditaan siirrettävyys kokonaan eri alustoiden (esim. xVM $\leftrightarrow$ Xen) välillä. Jälkimmäinen oleellisesti vaatii, että virtuaalipalvelimet tekevät käynnistyksen yhteydessä aina laitteiston tunnistuksen eivätkä sisällä ohjelmistoja, joiden toiminta on sidottu tiettyihin laitteisiin [37].

4.4 Verkon suunnittelu

Organisaation verkon suunnittelussa tarvitaan tietämystä sekä organisaation tietojärjestelmistä että verkkotekniikasta yleensä. Suunnittelua ei pidä jättää yhden henkilön tehtäväksi, vaan asiantuntevan tiimin on yhdessä päätettävä perusratkaisuista, jotka kestävät muuttumattomana vuosia eteenpäin. Näiden perusasioiden muuttaminen jälkikäteen on yleensä vaikea ja kallis prosessi, joten suunnittelu kannattaa tehdä kerralla kunnolla. Suunnittelussa tulee noudattaa joitakin perusperiaatteita [38]:

1. Suunnittelun lähtökohtana on käyttötarpeet, ei esimerkiksi vallitseva organisaatiorakenne.
2. Yksinkertaisin järkevä ratkaisu on paras – monimutkaisempia ratkaisuja voidaan harkita vain vahvojen perusteiden nojalla.
3. Suunnittelussa joudutaan aina tekemään kompromisseja. Näistä yleisimpiä on valinta edullisen hinnan, tehokkuuden ja saatavuuden välillä, joista voidaan käytännössä valita vain kaksi.
4. Kerran päätettyä perusratkaisua tulee noudattaa. Esimerkiksi uuden sovelluksen saamiseksi nopeasti käyttöön ei saa tehdä “pikakorjausta”, joka poikkeaa alkuperäisestä suunnitelmasta.

Sovelluskehityksen vaatimuksia verkolle ovat helppo konfiguroitavuus ja joustavuus sekä toisaalta tiukat palomuuriasetukset esimerkiksi tuotannon ja testiverkon välillä. Testiverkkoja voidaan edelleen erottaa toisistaan esimerkiksi IP-aliverkkoja (*subnet*) käyttämällä, mutta edellä listattua periaatetta (2) noudattaen tähän tulisi mennä vain, mikäli perusteet ovat vahvat. Monimutkaisia alitoimialueita, reitityksiä ja palomuurisääntöjä ei myöskään kannata käyttää, ellei niille todella ole tarvetta.

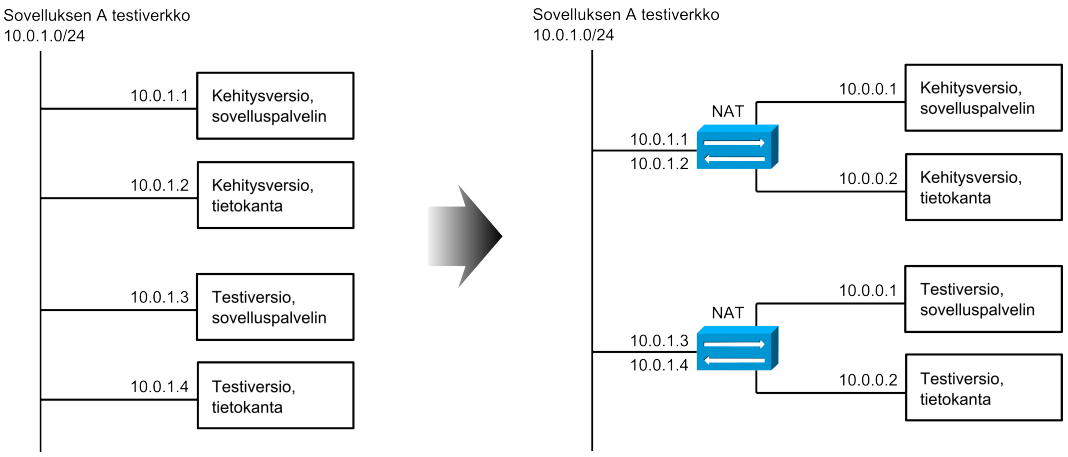
Tietoturvavaatimukset kuitenkin monimutkaistavat suunnittelua ympäristössä, jossa usean organisaation työtekijöitä toimii samassa verkossa ja samojen palvelimien parissa. Verkon turvallisuuspolitiikan suunnittelu etenee seuraavasti [39]:

1. Määritetään, minkälaisia riskejä organisaation verkossa olevaan dataan ja palveluihin voi kohdistua
2. Määritetään järjestelmän haavoittuvuudet eri riskejä vastaan
3. Analysoidaan verkon käyttäjien tarpeet
4. Laaditaan tietoturvapolitiikka, joka huomioi riskit, haavoittuvuudet ja käyttäjien tarpeet
5. Laaditaan toteutussuunnitelma sisältäen myös proseduurit, jotka täytyy käydä läpi aina tulevien muutosten yhteydessä

Sovelluskehitysympäristöissä yleisimmin toteutuvia riskejä ovat inhimilliset virheet, esimerkiksi väärän tietokannan käyttäminen tai tietyn komennon ajaminen väärässä järjestelmässä – pahimmillaan tuotantopalvelimella testipalvelimen asemesta. Tiedon joutuminen vääriin käsiin on myös toki organisaation kannalta vaarallista: esimerkiksi liikesalaisuudet tai työntekijöiden henkilötiedot eivät saa päätyä ulkopuolisille, eikä sovelluskehitystä tekevä ohjelmistoyrityskään halua vuotaa lähdekoodeja toiselle ohjelmistoyritykselle. Järjestelmän haavoittuvuudet liittyvät usein löysiin tietoturvakäytäntöihin – sovelluskehittäjät voivat käsitellä yhtä helposti niin tuotannon kuin testiympäristönkin dataa ja asetuksia, eikä testiverkkoja ole asianmukaisesti eristetty tuotantoverkosta.

Virtualisointi voi luoda tarpeen lisätä yksi ylimääräinen taso verkkorakenteeseen: kun palvelimia halutaan siirtää sovellustoimittajan ympäristöstä organisaation omaan verkkoon tai jopa yhden verkon sisällä testitasolta toiselle, siirtoja helpottaisi merkittävästi, jos palvelimien konfiguratataatioita ei tarvitsisi muuttaa lainkaan. Verkkokonfiguroinnin tarpeen voi eliminoida käyttämällä NAT-rajoitettuja aliverkkoja (kuva 14). Kuvan esimerkissä oikealla puolella sovelluspalvelin voi viitata tietokantaan aina osoitteella 10.0.0.2, tehtiinpä siitä kopio testitarkoituksiin tai olipa se toimittajan tai asiakkaan verkossa. VMware StageManagerilla pystyy automatisoidusti luomaan palvelimia tai palvelinryppäitä varten virtuaalireitittimiä, jotka suorittavat 1:1-NAT-operaation[11] verkkojen välillä [35].

[11] Jokainen sisäverkon osoite vastaa täsmälleen yhtä ulkoverkon osoitetta



Kuva 14: Palvelinkokonaisuuksien siirtäminen aliverkkoihin, joissa voidaan käyttää keskenään samoja IP-avaruuksia

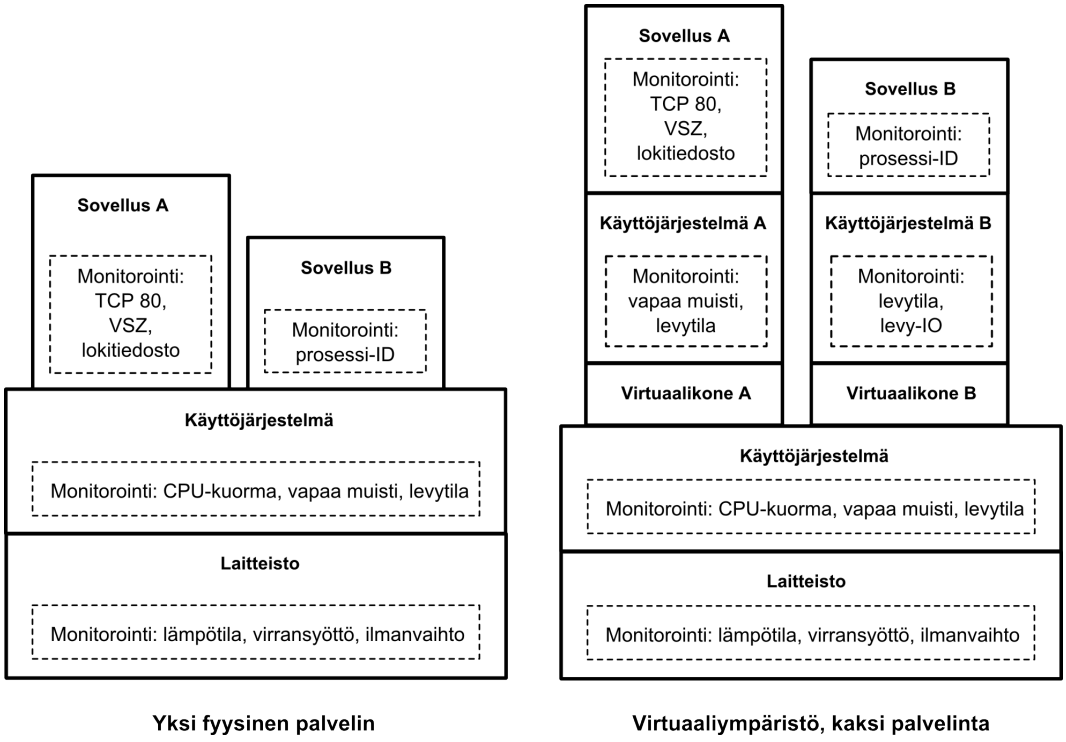
4.5 Palvelimien monitorointi

Tuotantoympäristön palvelimia on monitoroitava jatkuvasti ja automatisoidusti, jotta vikatilanteeseen voidaan reagoida välittömästi. Myös testiympäristön monitorointi on hyödyllistä: ei siksi, että palvelimilta edellytettäisiin jatkuvaa saatavuutta, vaan koska monitoroinnilla voidaan seurata helposti muistinkulutusta, prosessorikuormaa ja muiden resurssien käyttöä sovelluksia testattaessa. Erityisesti kuormitustestauksessa monitorointi on välttämätöntä statistiikkadatan keräämiseksi.

Perinteisesti monitorointia on tehty palvelimella ajettavilla monitorointityökaluilla, kuten Nagios, Munin, Simple Network Management Protocol -agentit (*SNMP*) sekä valmistajakohtaiset ratkaisut, joilla saadaan yksityiskohtaista tietoa laitteistosta. Monitorointi voidaan kohdistaa käytännössä mihin tahansa palvelimen ominaisuuteen, kuten fyysiseen laitteistoon (lämpötila, virransyöttö, levyjärjestelmä, ilmanvaihto), käyttöjärjestelmän resursseihin (prosessorikapasiteetti, muisti, levytila, verkon käyttö) tai sovelluksiin (prosessin tila, lokitiedoston sisältö, haluttu vastaus HTTP-kyselyyn tietyssä portissa).

Virtualisointi tuo myös monitorointiin uuden kerroksen: nyt voidaan suoraan isäntäkoneelta tarkkailla, kuinka paljon kukin virtuaalikone varaa resursseja. Esimerkiksi VMware ESX tarjoaa virtuaalikoneita varten SNMP-rajapintaan joukon päätason Management Information Base -rakenteita (*MIB*), jotka sisältävät tietoa mm. ajossa olevista virtuaalikoneista [40]. Tällä voidaan jossain määrin korvata käyttöjärjestelmätason monitorointi, mutta sovellustason tietoa isäntäkoneen kautta ei kuitenkaan saada. Isäntäkoneen resursseja, kuten prosessorikuormaa ja vapaata muistia, pitää tarkkailla aktiivisesti, koska ne vaikuttavat kaikkien virtuaalikoneiden toimintaan. Xen mahdollistaa virtuaalikoneiden tilatiedon tarkkailun isäntäkoneelta käsin käyttäen XenStore-järjestelmää [www-38].

Palvelimien monitoroinnin isäntäkoneen kautta tarjoama hyöty on kyseenalainen, koska useimmiten joudutaan sen lisäksi kuitenkin ajamaan kullakin virtuaalikoneella omaa monitorointityökalua sovellusten tarkkailua varten [40]. Ongelmalliseksi tilanteen tekee, että virtuaalipalvelimien sisältä ei välttämättä saadakaan korrektia tietoa laitteistosta – esimerkiksi prosessorikuorma voi näyttää hyvin kevyelläkin käytöllä suurelta, koska isäntäkone vaihtelee järjestelmän käytössä olevaa maksimiprosessoritehoa dynaamisesti. Tämä on suuri ongelma esimerkiksi kuormitustestauksessa.



Kuva 15: Monitoroinnin tasot fyysisessä ja virtualisoidussa ympäristössä, sekä esimerkit monitoroitavista kohteista

Virtualisointi siis helpottaa fyysisen laitteiston monitorointia, jota ei tarvitse tehdä kuin isäntäkoneelle, mutta monimutkaistaa virtualisoitujen resurssien tarkkailua. Esimerkiksi prosessorikuormaa, muistia ja levytilaa joudutaan nyt tarkkailemaan sekä isäntäkoneella että erikseen jokaisella virtuaalikoneella. Sovellustason monitorointiin virtualisointi ei vaikuta. Monitoroinnin eroavaisuuksia fyysisessä ja virtuaaliympäristössä selventää kuva 15.

Passiivisen monitoroinnin lisäksi kriittisille palveluille voidaan asettaa automaattisesti tapahtuvia toimenpiteitä, jotka laukeavat tietyn tapahtuman seurauksena. Esimerkiksi httpd-prosessin kuollessa HTTP-palvelin Apache käynnistetään välittömästi ja automaattisesti uudelleen. Linux-ympäristössä tähän voidaan käyttää esimerkiksi Monit-työkalua [www-39]. Klusteroidussa ympäristössä on mahdollista automatisoida koko palvelimen uudelleenkäynnistyminen tietyn virhetilanteen seurauksena.

4.5.1 Lista tarkkailtavista ominaisuuksista

Laitteistotasolla tarkkailumahdollisuudet asettavat palvelinalusta ja palvelinvalmistajan tarkkailuohjelmisto. Tarkkailun piirissä voivat olla:

- Prosessorin, emolevyn, kotelon, kovalevyjen, virtalähteen, muistin sekä huoneilman lämpötilat
- Levyjärjestelmän toiminta sekä yksittäisten levyjen tilanne (S.M.A.R.T [41])
- Virransyötön tila, varavirtajärjestelmän tila
- Kotelon sisäinen ilmanvaihto sekä räkin tai palvelintilan ilmastointi
- Lisälaitteiden, kuten nauhavarmistimen tai konsolikortin toiminta

Käyttöjärjestelmätason resursseista voidaan monitoroida mm.:

- Prosessorin kuorma sekä kuorman jakautuminen eri prosessien kesken
- Vapaan muistin määrä sekä yksittäisten prosessien muistinkäyttö
- Vapaa levytila, tiedostojen määrä
- IO-kuorma, erityisesti levy-IO sekä verkko-IO
- Prosessien kokonaismäärä sekä eri tiloissa olevien prosessien määrä (mm. zombie-prosessit ja aktiiviset prosessit)
- Avoimien verkkoyhteyksien määrä
- Käyttäjien määrä

Sovellustason monitorointimahdollisuudet riippuvat täysin sovelluksesta, mutta joitakin yleisiä parametrejä voidaan tarkkailla lähes kaikilta sovelluksilta:

- Onko prosessi käynnissä
- Prosessien tai säikeiden määrä
- Vastaus verkon kautta tulevaan kyselyyn
- Muistin käyttö
- Käyttäjäistuntojen määrä

4.6 Varmuuskopiointi ja arkistointi

Virtuaalikoneet mahdollistavat tavallisen käyttöjärjestelmätason tai sovellustason varmuuskopioinnin lisäksi koko koneen varmuuskopioimisen suoraan levytasolla. Useimmissa virtuaaliympäristöissä tämä onnistuu myös virtuaalikoneen ollessa päällä ottamalla koneesta tilannekuva ja kopioimalla alkuperäiset levykuvat sekä konfiguraatiotiedostot talteen. Lopuksi ajossa olevan δ-levyn voi konsolidoida takaisin varsinaiseen levyyn (kpl 3.5). Molemmilla varmuuskopiointitavoilla on omat käyttökohteensa.

Palvelimen tiedostojen varmuuskopiointi tiedostotasolla jonkin varmuuskopiointiohjelmiston agenttia käyttäen tai sovelluksen datan varmistaminen sovelluksen omalla työkalulla, esimerkiksi Oracle Recovery Managerilla (*rman*) [www-40], on yleensä nopeampaa ja joustavampaa kuin koko levykuvan varmuuskopioiminen. Differentiaalisesti tai inkrementaalisesti tehtynä varmuuskopioidaan päivittäin vain edellisen varmuuskopion jälkeen muuttunut data. Differentiaalinen varmuuskopio tarkoittaa muuttuneita tiedostoja edelliseen täyteen varmistukseen nähden, inkrementaalinen edelliseen inkrementtiin nähden; palautustilanteessa ensimmäisessä tapauksessa tarvitaan siis edellinen täysi kopio sekä viimeisin differentiaali, jälkimmäisessä tapauksessa edellinen täysi kopio sekä sen jälkeen tehdyt kaikki inkrementit. Joustavuus saavutetaan mahdollisuudella palauttaa mikä tahansa yksittäinenkin tiedosto halutun päivän versioon [42]. Useimmissa tilanteissa data on lisäksi palvelinriippumatonta: data voidaan palauttaa toiselle, eri arkkitehtuuria ja käyttöjärjestelmää käyttävälle palvelimelle, kunhan käytettävä ohjelmisto on yhteensopiva.

Tiedostotason varmuuskopion palauttaminen on kuitenkin hidasta esimerkiksi levyrikon jälkeen, koska järjestelmä täytyy asentaa toimintakuntoon ennen kuin tiedostoja pystytään palauttamaan. Jos varmuuskopio on vain sovelluksen datasta, myös ohjelmistot pitää asentaa ensin manuaalisesti. Näihin toimenpiteisiin kuluu helposti työpäivä tai jopa kaksi, joka on ilman muuta liian pitkä aika tuotantojärjestelmän katkokselle.

Virtuaaliympäristö tarjoaa tähän tilanteeseen helpon ratkaisun: kopioidaan koko levykuva blokkitasolla. Isäntäkoneen pettäessä levykuva voidaan palauttaa varmistukselta toiselle isäntäkoneelle ja käynnistää välittömästi – toimenpiteeseen kuluva aika riippuu levyn koosta, mutta kysymys on ennemmin minuuteista kuin tunneista.

4.6.1 Ajonaikainen varmuuskopio virtuaalikoneesta

Ajonaikaisen varmuuskopion tekeminen tiedostojärjestelmästä blokkitasolla on monimutkainen toimenpide ja sisältää merkittäviä riskejä. Tärkeää dataa voi olla kopiota tehtäessä muistissa, ja tämä data menetetään, mikäli muistin sisältöä ei kirjoiteta levylle ennen varmuuskopiointia. Muistin sisällön menetys voi johtaa tietokantapalvelimen tapauksessa jopa kannan joutumiseen niin epäkonsistenttiin tilaan, että tietokanta ei

enää palautuksen jälkeen lähde käyntiin. Muistin talteenottaminen sellaisenaan vaatisi käyttökatkoksen, joka ei sovi jatkuvassa käytössä oleville palvelimille. Sekä Microsoft (Volume Shadow Copy, VSS [43]) että VMware (VMware Consolidated Backup, VCB) tarjoavat samankanlaista prosessia ongelman ratkaisuksi[44]:

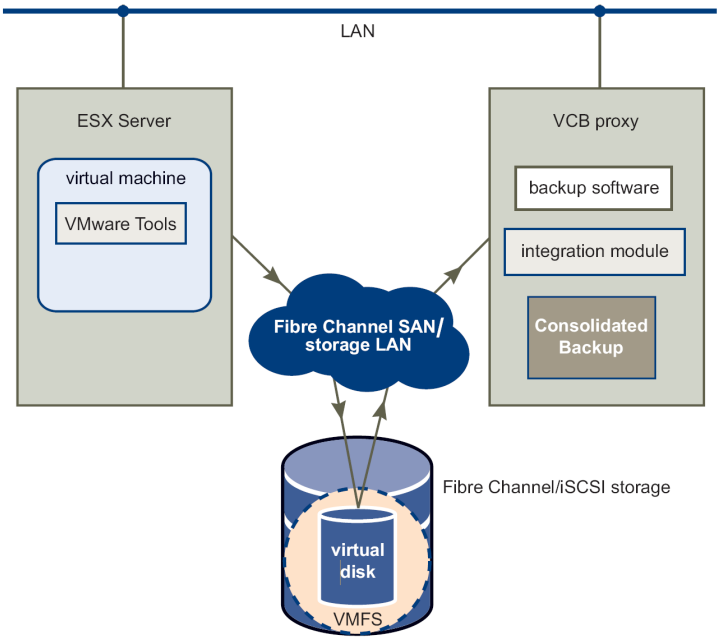
1. Laitetaan sovellukset varmuuskopiotilaan. Tällöin ne osaavat mm. tyhjentää omat välimuistinsa.
2. Tyhjennetään tiedostojärjestelmän puskurit ja asetetaan tiedostojärjestelmä pysäytettyyn tilaan.
3. Otetaan virtuaalikoneesta tilannekuva.
4. Palautetaan tiedostojärjestelmä tavalliseen tilaan, jolloin palvelin voi jatkaa normaalia toimintaa.
5. Palautetaan sovellukset tavalliseen toimintatilaan.
6. Tehdään tilannekuvasta varmuuskopio.
7. Konsolidoidaan δ-levy ja alkuperäinen levy.

Prosessiin sisältyy mahdollisia ongelmakohtia. Vaihe 1 vaatii, että sovellukset on suunniteltu varmuuskopiointi huomioonottaen. Vaiheissa 2–4 kuluu aikaa joistakin sekunneista joihinkin minuutteihin niin, että sovellukset eivät pysty vastaamaan pyyntöihin – tämä voi olla ongelma jatkuvassa käytössä olevilla palvelimilla. Vaiheiden 6–7 aikana levy-IO on huomattavasti hitaampaa δ-levyn käytön takia.

Windows-pohjaisia virtuaalikoneita ja NTFS-tiedostojärjestelmää käytettäessä VCB osaa tehdä levykuvasta tiedostotason varmuuskopion jopa differentiaalisena tai inkrementaalisena [44]. Lisäksi järjestelmä voidaan rakentaa niin, ettei varmuuskopiointi rasita isäntäkonetta tai edes verkkoa, vaan erillinen varmistuspalvelin (*VCB proxy*) hoitaa toimenpiteet suoraan levyjärjestelmän tasolla esimerkiksi Fibre Channelia käyttäen (kuva 16).

Saman asian toteuttaminen on teknisesti mahdollista myös käyttäen avoimen koodin työkaluja Xen-ympäristössä:

1. Tyhjennetään sovellusten puskurit ja välimuistit. Varmin tapa pitää puskurit tyhjänä on pysäyttää sovellukset.
2. Tyhjennetään tiedostojärjestelmän puskurit.
3. Tehdään virtuaalikoneen levystä tilannekuva LVM Snapshot -toiminnolla [31]. Tämä edellyttää, että virtuaalikoneet käyttävät LVM-pohjaisia asemia tiedostopohjaisten virtuaalilevyjen asemesta.



Kuva 16: Varmuuskopiointi Storage Area Networkia ja erillistä varmistuspalvelinta käyttäen [44]

4. Palautetaan sovellukset toimintaan.

5. Tehdään tilannekuvasta varmuuskopio. Tämä voidaan tehdä duplikoimalla koko LVM-asema tai lukemalla aseman sisältö tiedostotasolla.

6. Konsolidoidaan δ-levy ja alkuperäinen levy.

Ongelmat liittyvät tässäkin ratkaisussa vaiheeseen 1, jonka edellyttämänä palveluihin tulee lyhyt toimintakatkos.

4.7 Sovellustestaus

Testaus jaetaan karkeasti kahteen kategoriaan: funktionaaliseen testaukseen ja ei-funktionaaliseen testaukseen. Ensimmäisen tarkoitus on varmistaa, että sovellus vastaa sen toiminnallisia määrityksiä: sen täytyy sisältää vaaditut toiminnot, tarjota määritellyt rajapinnat ja suorittaa oikeat operaatiot. Jälkimmäiseen ryhmään kuuluvat muut kuin sovelluksen sisäisen toiminnan kannalta oleelliset asiat: muun muassa käyttäjän kokema suorituskyky, vakaus, käytettävyys ja ohjeistus [46]. Tietoturvan ja yhteensopivuuden testaus voi tilanteesta riippuen kuulua kumpaan tahansa kategoriaan.

Tämän työn puitteissa testauksessa kiinnostavaa on palvelin- ja kehitysympäristön vaikutus testauksen kulkuun.

4.7.1 Sovellusten testaus työasemilla

Työasemakohtaiset virtuaaliympäristöt helpottavat sekä funktionaalista että epäfunktionaalista testausta. Moderneilla, tehokkailla työasemilla saadaan jokaisen sovelluskehittäjän omaan ympäristöön niin tietokanta- kuin sovelluspalvelinkin. Palvelinkokonaisuuksien jakaminen tiimin kesken on erinomaisen helppoa esimerkiksi OVF-formaatissa (luku 4.3.1). 10-henkinen sovelluskehitystiimi säästää helposti jopa kymmeniä henkilötyöpäiviä niin, että vain yksi tiimistä asentaa tarvittavat konfiguraatiot ja jakaa ne muille kehittäjille virtuaalikoneina.

Testauksessa sovelluskehittäjä voi parhaimmillaan ajaa kahta sovellusversiota rinnakkain ja verrata näiden toimintaa. Funktionaaliset testit ja löydettyjen virheiden korjaaminen onnistuvat näppärästi, mutta myös alustavat stabiilius- ja suorituskykytestit on mahdollista suorittaa työasemilla. Omalla työasemalla toimittaessa muut käyttäjät eivät aiheuta suorituskykyvolatiliteettia tai häiritse järjestelmän toimintaa muilla testeillä, joten tulokset ovat keskenään vertailukelpoisia, vaikka toinen testi olisi ajettu päivällä ja toinen yöllä. Työasemalla ajetut testit eivät tietenkään kerro totuutta kuormituksen käyttäytymisestä sadoilla oikeilla käyttäjillä.

4.7.2 Suorituskykytestaus

Sovelluksen laadunvalvontaan kuuluu suorituskyvyn testaaminen ennen sovelluksen käyttöönottoa. Suorituskykytestauksella on kolme päätarkoitusta [47]:

1. Varmistaa, että sovellus tai tietty sen komponentti täyttää määritetyt suorituskykyvaatimukset
2. Mitata sovelluksen maksimikapasiteetti sekä kuormitusraja-arvot
3. Löytää mahdolliset muut suorituskykyongelmat, pullonkaulat ja optimointitarpeet

Suorituskykytestaus mittaa muun muassa seuraavia asioita: toimintojen nopeus ja latenssit, käyttäjän kokema vaste-aika; prosessorin, muistin, levyjärjestelmän ja verkon kuormitusaste; transaktioiden läpäisykyky; luotettavuus ja saatavuus; sekä skaalautuvuus eri käyttäjämäärille ja kuormitustilanteisiin. Lähes kaikille attribuuteille määritetään minimi-, keski- ja maksimiarvot [47].

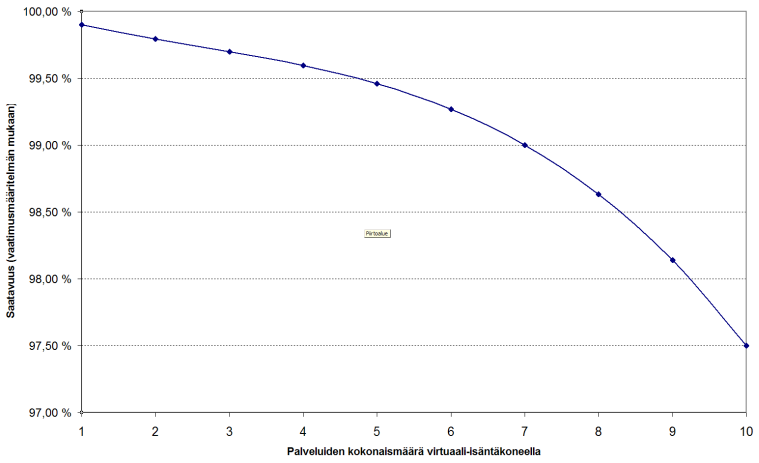
Kaikki edelliset riippuvat koko palvelinympäristöstä, mukaan lukien itse sovellus, palvelinohjelmisto, palvelinkone, taustapalvelut, verkko, levyjärjestelmä ja muiden sovellusten aiheuttama kuorma. Lisäksi järjestelmän suorituskyky vaihtelee dynaamisesti mm. kellonajan ja viikonpäivän mukaan. Satunnaisia, suuria kuormituspiikkejä voivat aiheuttaa esimerkiksi intra-järjestelmässä julkaistavat koko organisaatiota koskevat uutiset tai

työasemien automaattiset päivitykset. Näiden kaikkien takia on lähes mahdotonta luoda täysin tuotantoympäristöä vastaavaa testitilannetta, mikä pakottaa tekemään arvioita testin ja tuotannon korrelaatiosta. Laadunvalvonnan yleisten periaatteiden mukaan onkin tarkoitus saavuttaa *riittävän hyvä varmuus*, että testattu suorituskyky vastaa todellista tuotannon suorituskykyä.

Virtualisointi lisää testitulosten hajontaa: sovelluksen suorituskyky alkaa vaihdella myös muiden sovellusten käyttöasteen mukaan, vaikka ne eivät toimisikaan taustapalveluina testattavalle sovellukselle. Yöllä ajettujen testien tulokset poikkeavat entistä enemmän todellisesta, koska yöllä koko virtuaalijärjestelmän kuorma on pieni.

Sovellustoimittajan tehtävä on kuitenkin vain varmistaa, että sovellus täyttää suorituskykykriteerit vaatimusmäärityksissä esitetyllä kokoonpanolla. Voidaan esimerkiksi määritellä, että sovelluksen tulee kestää 500 käyttäjän kuorma palvelimella, jossa on kaksi 2.0 GHz prosessoria ja 4 GiB muistia. Testaaja luo tällöin virtuaaliympäristöön kyseisen kokoonpanon ja ajaa testit yöllä. Virtuaalipalvelinjärjestelmän ylläpitäjän tehtävä puolestaan on varmistaa, että tuotantoympäristössä on vähintään yhtä paljon resursseja käytettävissä vielä muiden sovellusten ja käyttäjien lisäksi.

Sallittavan riskin puitteissa virtuaalijärjestelmän resursseja voidaan ylivarata niin, että sovellukset toimivat riittävän suurella varmuudella, mutta käyttöpiikkien aikana on mahdollista, että vasteajat nousevat yli toleranssien tai kaikki käyttäjät eivät mahdu järjestelmään sisään. Tällä tavoin voidaan huomattavasti parantaa laitteistoresurssien käyttöastetta saatavuuden kustannuksella (kuva 17). Sovellukset voivat kuitenkin käyttäytyä ylivaraustilanteissa odottamattomasti, koska pullonkaulat siirtyvät todennäköisesti eri paikkoihin kuin sovelluskehittäjien suorituskykytesteissä.



Kuva 17: Sovelluksen saatavuus virtuaaliympäristön kuormituksesta riippuen (esimerkki)

Vastuun jako ongelmatilanteissa muuttuu vaikeaksi: johtuvatko suorituskykyongelmat sovelluksesta vai virtuaaliympäristöstä? Resursseja periaatteessa on vaatimusmäärityk-

sien mukainen määrä, mutta niitä voivat käyttää muutkin sovellukset yhtä aikaa. Ongelmia voidaan ennaltaehkäistä suorittamalla suorituskykytestaus tuotantoympäristössä tai muussa ympäristössä, jossa taustakuormitus vastaa tuotantoa. Hyödyllisten tulosten saamiseksi testaus pitää suorittaa kuormitushuipun aikaan, mutta tästä on taas haittana ympäristön muiden sovellusten käytettävyyden heikkeneminen.

4.8 Yhteenveto

Virtualisointi sekä tehostaa että monimutkaistaa sovelluskehitysprosessia. Virtualisointikerroksen lisääminen helpottaa palvelimien hallintaa, mahdollistaen täysin uusia tapoja sovellusten testaamiseen ja jakeluun. Esimerkiksi tilannekuvien käyttäminen ja useiden palvelimien ajaminen rinnakkain helpottavat testaajan työtä suuresti. Palvelinkokonaisuuksien jakaminen niin kollegoille testaukseen kuin asiakkaille tuotantoa varten nopeutuvat Open Virtualization Formatia (OVF) käyttäen.

Toisaalta virtualisointi hankaloittaa suorituskyvyn ja resurssitarpeen arvointia. Testiympäristössä suorituskykytestin läpäissyt sovellus voi aiheuttaa tuotantoympäristöön niin suuren kuormituksen, että muiden samalla virtuaali-isäntäkoneella toimivien sovellusten käytettävyys laskee alle toleranssien.

Virtualisointi mahdollistaa palvelinkokonaisuuden kuljettamisen koko sovelluksen elinkaaren läpi – kehitysvaiheista tuotantoon ja lopuksi arkistoon – mutta palvelinkonfiguraation muutostarpeet eri vaiheiden välillä aiheuttavat ongelmia. Esimerkiksi testauksen ja tuotannon välillä palvelimeen pitäisi muuttaa kaikki tietoturva- ja käyttäjäasetukset. Niinpä sovelluksen julkaisunhallinta voi olla helpompi toteuttaa edelleen perinteisesti asentamalla sovellus eri vaiheissa eri palvelimille kuin käyttämällä samaa palvelinta koko elinkaaren läpi.

5 Use case: Organisaation sovelluskehitys- ja palvelinympäristön virtualisointi

Tekes – teknologian ja innovaatioiden kehittämiskeskus (myöh. Tekes) edustaa tietojärjestelmien kannalta modernia organisaatiota, jossa on käytössä useita organisaation sisäisiä järjestelmiä, joitakin yhteistyöorganisaatiolle tarjottavia palveluita sekä asiakkaille näkyvä julkinen asiointipalvelu. Sisäisiä järjestelmiä, erityisesti rahoituksentinhallintaa ("Sovellus A") sekä intranet-portaalia käyttää suurin osa organisaation työntekijöistä, reilut 300 henkeä 400:stä. Julkinen verkkoasiointipalvelu on vasta hiljattain otettu käyttöön, joten sen käyttäjämäärä on vielä vähäinen, mutta nousee tasaisesti. Ensimmäisen kahdeksan kuukauden aikana verkkoasiointiin rekisteröityi noin tuhat käyttäjää. Julkisen palvelun potentiaalisia käyttäjiä ovat kaikki Tekesin asiakkaat.

5.1 Järjestelmäarkkitehtuuri

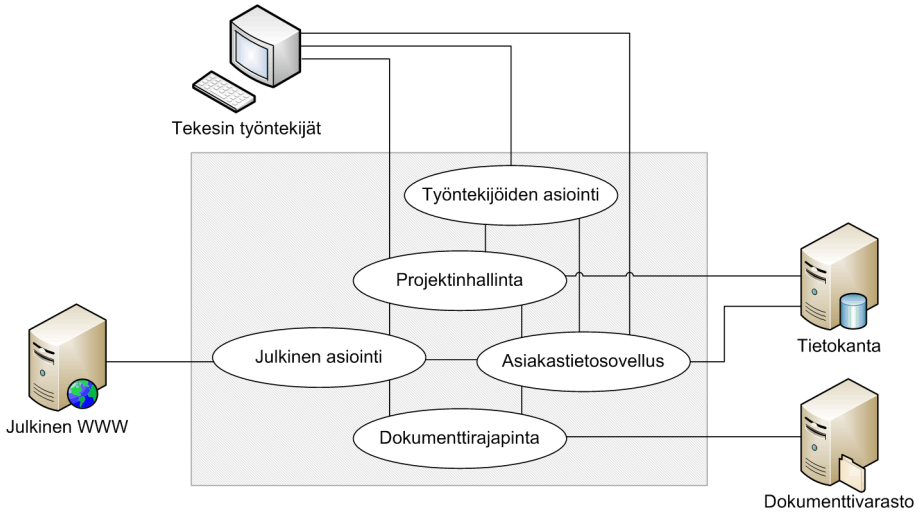
Tämän hetken vallitseva suuntaus organisaatiotietojärjestelmissä on palvelusuuntautunut arkkitehtuuri eli Service-Oriented Architecture (SOA). Olennaista arkkitehtuurissa on, että organisaation tietojärjestelmät koostuvat useista erillisistä järjestelmistä, jotka liitetään toisiinsa yhteisten rajapintojen kautta. Näin rakennettuja modulaarisia järjestelmiä pystytään hallitsemaan ja ylläpitämään komponenteittain, edellyttäen, että sovellukset tukevat yhteisiä rajapintoja.

Tekesin tietojärjestelmiä on kehitetty SOA-suuntaan käyttämällä tiettyyn tarkoitukseen tehtyjä sovelluksia, jotka pystyvät tarjoamaan dataa joko suoraan käyttäjille tai jonkin rajapinnan läpi muille sovelluksille. J2EE-sovellukset käyttävät Enterprise Java Beans (EJB)-arkkitehtuuria [www-41], ja muiden sovellusten välillä tietoa kuljetetaan mm. Web Services-rajapinnan [www-42] läpi. Esimerkiksi asiakastietoja ja projektitietoja tarvitaan useassa eri sovelluksessa, mutta ei ole järkevää, että näitä käsittelevä logiikka sisällytettäisiin kaikkiin sovelluksiin erikseen. Kuvassa 18 on esitetty muutamien Tekesin ydinsovellusten riippuvuussuhteet ja liitynnät muihin sovelluksiin.

5.2 Palvelin- ja verkkoarkkitehtuuri

Käytössä on tällä hetkellä kirjava joukko alustoja eri palveluillle. Ympäristöstä löytyy mm. eri versioita Windowsista ja Linux-jakeluista, 32- ja 64-bittisiä järjestelmiä, eri valmistajien tietokantoja, lukuisa joukko eri rajapintoja sovellusten välillä, erilaisia ylläpito- ja asennuskäytäntöjä sekä palvelintasolla niin fyysisiä kuin virtuaalisia koneita ja jopa eri prosessoriarkkitehtuureja.

Suuri diversiteetti vaikeuttaa paljon järjestelmien ylläpitämistä. Monesti joudutaan konsultoimaan järjestelmän asentanutta henkilöä pienissäkin asioissa; esimerkiksi käyttäjän



Kuva 18: Osa Tekesin yhteen integroiduista sovelluksista

lisäys eri järjestelmiin tapahtuu lähes aina eri tavalla. Tekesillä onkin linjaus monien asioiden standardointiin. Linjauksista poikkeavia nk. pysyviä väliaikaisratkaisuja tulee kuitenkin rakennettua liian helposti, ja näiden muuttaminen standardinmukaisiksi järjestelmiksi jää hyvin usein tekemättä. Kun uusi järjestelmä on saatu toimimaan, siihen ei haluta kajota, ellei toiminnassa ilmene vikaa. Tämän takia pitäisi heti alkumetreillä toimia suunniteltujen käytäntöjen mukaan, niin jälkikorjauksia ei jouduttaisi tekemään. Toisaalta käytäntöjen pitää olla sen verran joustavia ja hyvin suunniteltuja, että niitä voidaan noudattaa ilman ylivoimaisia ponnisteluja tai kompromisseja.

Esimerkiksi uuden tietojärjestelmän "B" tuominen sovellustoimittajalta Tekesille tapahtui tehokkaasti ns. virtual-applianceina (ks. 4.3.1). Toimitukseen kuului myös tietokantapalvelin virtuaalikoneena. Näin toimitettu järjestelmä kyllä toimi ongelmitta, mutta tietokanta ei tullut mukaan keskitettyyn hallintaan muiden tietokantojen kanssa (mm. varmuuskopioiden ja tarkkailun piiriin), ja siitä olisi pitänyt maksaa uusi kallis lisenssi. Tietokanta olisi haluttu asentaa vanhalle tietokantapalvelimelle, mutta sen siirtäminen jälkikäteen osoittautui yllättävän haasteelliseksi. Jos tietokanta olisi jo alun periin luotu stardardikäytäntöjen mukaan, ongelmilta olisi vältytty kokonaan.

5.3 Virtualisointiin ajavat tekijät

Tekesillä virtuaaliympäristöön siirtyminen nousi vakavasti harkittavaksi projektiksi usean osatekijän summana. Koko ympäristöä ei ole tarkoitus muuttaa virtuaaliseksi kerralla, vaan lähtien testipalvelimista ja siirtyen tuotantopalvelimiin sitä mukaa, kun palvelimia jouduttaisiin korvaamaan uusilla. Joidenkin järjestelmien osalta aikataulua haluttiin nopeuttaa, koska virtualisointi helpotti merkittävästi näiden ylläpitoa.

Tekesin liiketoiminnan kannalta tärkeät sovellukset ovat kasvaneet suuriksi ja monimutkaisiksi systeemeiksi, joilla on useita toimittajia ja joiden testausta ja kehitystä tekevät

monet tahot yhtä aikaa eri vaiheissa. Suurimpiin tuotantojärjestelmiin kuuluu suoraan viisi, jopa kuusi palvelinkonetta järjestelmää kohden, ja järjestelmien kehitys ja testaus vaatii ainakin yhtä monta lisää. Tällaisen palvelinmäärän ylläpitäminen fyysisinä koneina on yksinkertaisesti liian kallista. Jos toimittaisiin vain fyysisillä koneilla, kehitystä ja testausta pitäisi rajoittaa.

Virtualisoinnilla saadaan palvelimien resurssit huomattavasti tehokkaammin hyödynnettyä. Testauksessa ja kehityksessä keskimääräinen kuormitus on hyvin vähäistä tai satunnaista, jolloin samalla fyysisellä koneella voidaan huoletta ajaa useaa ympäristöä. Tietyn järjestelmän testauksen tai kehityksen päätyttyä virtuaalipalvelinresurssit on helppo ottaa seuraavan projektin käyttöön – palvelinkoneet voidaan tuhota tai arkistoida napin painalluksella, minkä jälkeen resurssit ovat heti uusiokäytettävissä.

Virtualisoinnin tuomia mahdollisuuksia halutaan myös hyödyntää manuaalisen työn vähentämiseksi. Testiympäristöjen luomiseen käytetään kloonausta ja pohjia (*templates*), joiden avulla viiden uuden palvelimen asentaminen onnistuu parissa tunnissa verrattuna muutamaan työpäivään, joka kuluisi fyysisten palvelimien kanssa. Testauksessa kehittäjät käyttävät tilannekuvia, joiden avulla voivat nopeasti palata tilanteeseen ennen tiettyä operaatiota – perinteisesti järjestelmästä olisi pitänyt ottaa varmuuskopio ennen operaatiota. Tilannekuvilla on saatu jo aikaan todella suurta säästöä työtunneissa: noin kymmenestä palvelimesta olisi pitänyt ottaa täysi varmuuskopio ennen riskialttiin päivityksen asentamista, mutta varmistusten sijaan tehtiin tilannekuvat muutamassa minuutissa. Toisessa tapauksessa tietokannan siirto testipalvelimelta toiselle epäonnistui hävittäen osan datasta, mutta ennen operaatiota tehtyyn tilannekuvaan palaaminen onnistui noin kymmenessä minuutissa, kun täyteen uudelleenasennukseen olisi mennyt vähintään kokonainen työpäivä.

5.4 Uuden ympäristön vaatimukset ja toivotut ominaisuudet

Ympäristön tulee olla ilman muuta riittävän tehokas ja vakaa. Erityisesti virtuaaliympäristöltä odotetaan vanhaan ympäristöön verrattuna parempaa skaalautuvuutta, helppokäyttöisyyttä ja ylläpidettävyyttä, ja näiden myötä parempaa kustannustehokkuutta. Ympäristön pitää olla tietoturvallinen ja tuotantojärjestelmän osalta saatavuus (*availability*) pitää olla vähintään yhtä korkea kuin fyysisillä palvelimilla.

5.4.1 Tehokkuus

Tuotantojärjestelmien palvelujen suorituskyvyssä ei saa tapahtua hetkellisiäkään notkahduksia alle toleranssirajojen. Tämän myötä virtuaalialustan resursseja ei voida ylivarata (ks. 3.5) juuri lainkaan, vaan muistia ja prosessoritehoa on pidettävä vapaana

äkillisiä kuormitustilanteita varten. Tämä laskee resurssien käyttöastetta selvästi, mutta on välttämätöntä jouhevan toiminnan takaamiseksi.

Tekesin testiympäristössä voidaan olettaa, että järjestelmiä A, B ja C ei kovin suurella todennäköisyydellä kuormiteta yhtä aikaa, joten mitoitetaan resurssit paljon tiukemmin. Palveluiden suorituskyky saattaa vaihdella, mutta siitä ei koidu suurta vahinkoa. Käytännössä eniten rajoittaviksi tekijöiksi on havaittu sovellusten toiminnan kannalta muistin ja levytilan määrä, sekä muissa operaatioissa usein levykirjoitusten hitaus. Muistin määrään saatiin nopea ratkaisu yksinkertaisesti ostamalla sitä lisää, mutta levyjärjestelmän tilanne on haastavampi: enterprise-järjestelmät ovat kalliita ja myös niiden käyttämät levyt ovat kalliita. Levykirjoitusten nopeus ei ole suoraan verrannollinen levyjen nopeuteen, vaan virtuaalipalvelimien tapa käyttää levyjä on usein merkityksellisempää.

Kaikkien palveluiden osalta toivomus ja osin vaatimuskin on, että virtualisoidun järjestelmän suorituskyky ei saa olla heikompi kuin edeltäneen fyysisen järjestelmän. Yleensä tämä ei ole ongelma, koska virtuaali-isäntäpalvelimet toimivat uudella, tehokkaalla laitteistolla, kun vanhat palvelut ovat edellisen sukupolven koneilla.

5.4.2 Skaalautuvuus

Virtuaaliympäristö voitaisiin toteuttaa ilman klusterointia käyttämällä yhtä erittäin järeää x86-palvelinta, jota tällä hetkellä edustaa esim. Unisys ES7000 Model 7600R. Sen kuhunkin 16 prosessorikantaan voidaan asentaa 6-ytiminen Intel Xeon -prosessori, joilla päästään yhteensä 96 prosessoriytimeen. Muuta kuin x86-tekniikkaa käyttäen on saatavilla paljon suurempiakin palvelimia, mutta ne eivät käytännössä sovellu sekalaisten käyttöjärjestelmien ja sovellusten ajamiseen.

Yksittäisen koneen skaalautuvuus on kuitenkin heikko. Vaikka koneessa olisikin vapaita prosessori- ja muistikantoja, ei näitä voida lisätä konetta sammuttamatta, mikä vaatisi kaikkien niiden kymmenien tai jopa satojen koneella ajettavien virtuaalikoneiden sammuttamisen. Suuri riski liittyy myös siihen, että koneen vikaantuessa kaikki virtuaalipalvelimet, ja sitä myötä kaikki organisaation tietojärjestelmät, putoavat pois käytöstä.

Klusterien skaalautuvuus ja vikasietoisuus ovat paljon parempia, mutta niissä resurssien jakaminen edellyttää enemmän järjestelyjä. Virtuaalikoneet pitäisi sijoittaa resurssien käytön mukaan tasaisesti eri isäntäkoneille, mutta jatkuvasti muuttuvien tarpeiden takia se ei ole helppoa. Virtuaalialustojen LiveMigration-järjestelmä (ks. 3.4) huolehtii palvelimien automaattisesta tasaamisesta virtuaali-isäntien kesken, mutta ominaisuudesta joutuu maksamaan korkeamman lisenssihinnan.

iSCSI:iin perustuva levyjärjestelmä on virtuaaliklusterin ulkopuoleinen ratkaisu, ja sitä voidaan käyttää myös fyysisten palvelimien kanssa. Levyjärjestelmän täytyy voida skaalautua virtuaalijärjestelmän mukana.

5.4.3 Helppokäyttöisyys

Perustoimintojen, kuten palvelimien kloonaus, konfigurointi, tuhoaminen, siirtäminen toiselle isännälle ja levyjen lisääminen tulee olla helppoja, nopeita ja riskittömiä. Tämä ei ole itsestäänselvyys kaikilla alustoilla: esimerkiksi Xenissä palvelimen kloonaus edellyttää sekä levykuvan että konfiguraatiotiedoston manuaalista kopiointia sekä jälkimmäisen muuttamista käsin. Suurin osa graafisista hallintaliittymistä tarjoaa riittävän hyvät perustoiminnot.

Helppokäyttöisyyden piiriin voidaan laskea myös Open Virtualization Format -tuki, jonka avulla voidaan siirtää valmiita palvelinkokonaisuuksia toimittajalta organisaatioon ja päinvastoin. Ilman OVF-tukea jouduttaisiin käyttämään erilaisia muuntimia (*converter*) virtuaalikoneiden tuomiseen vähänkin poikkeavasta ympäristöstä.

5.4.4 Ylläpidettävyys

Helppokäyttöisyys ja skaalautuvuus ovat jo osa ylläpidettävyyttä, mutta lisäksi täytyy ottaa huomioon varmuuskopiointi, päivitykset ja käyttöoikeuksien hallinta. Varmuuskopioinnissa olisi suuri askel eteenpäin, jos selvittäisiin ilman palvelimen sisäisiä varmuuskopiointiagentteja (ks. 4.6). Agenttien asentaminen ja konfigurointi eri käyttöjärjestelmiin on työlästä, ja palvelimen kloonauksen jälkeen voidaan päätyä sekavaan tilanteeseen, kun kaksi eri palvelinta koettaa tehdä varmuuskopiota samoilla agentin asetuksilla.

Virtuaalialustan tietoturvapäivitysten läpivientiin pitää olla joustava ratkaisu. Klusteroitu ympäristö mahdollistaa virtuaalipalvelimien siirtämisen pois tietyltä isäntäkoneelta päivityksen ajaksi, mutta palvelimien käyttöasteen pitää olla riittävän alhainen, että yksi isäntäkone voidaan ottaa hetkellisesti kokonaan pois käytöstä.

Käyttöoikeuksien hallinnalla voidaan sallia esimerkiksi sovelluskehittäjien tehdä muutoksia määrättyjen palvelimien konfiguraatioon. Tällä voidaan vähentää järjestelmänvalvojan työtaakkaa ja nopeuttaa yksinkertaisia toimenpiteitä, kuten muistin tai kovalevyn lisääminen. Sallittuja toimenpiteitä pitää voida rajoittaa tarpeeksi tehokkaasti, että muutoksilla ei voida häiritä muiden koneiden toimintaa.

5.5 Tekesin virtuaaliympäristö

Tekesillä päädyttiin VMware ESX -virtuaalialustaan, joka asennettiin korttipalvelimista (*blade server*) koostuvaan klusteriin. Tuotantoverkkoon otettiin käyttöön LiveMigration-ominaisuus kuorman automaattiseksi tasaamiseksi isäntäkoneiden välillä, mutta testiverkkoon ominaisuuden ei katsottu tarjoavan lisähinnan oikeuttavaa etua.

VMware tarjoaa myös nimenomaan sovelluskehitykseen tarkoitettua Stage Manager -tuotetta, joka mahdollistaa palvelimien ja palvelinryppäiden kuljettamisen eri kehitys- ja testaustasojen välillä web-käyttöliittymän kautta. Stage Manager osaa automatisoidusti luoda eristettyjä testiverkkoja sekä suorittaa itsenäisesti uusille palvelimille joitakin konfiguraatio-operaatioita, kuten IP-osoitteen vaihtaminen. Pilotoinnissa Stage Manager havaittiin kohtuullisen toimivaksi – palvelimien automaattinen konfigurointi, joka oli yksi kätevimpiä ominaisuuksia, tosin toimi vaihtelevasti. Tuotteen korkea hinta kuitenkin kallisti hankitapäätöksen kielteiseksi, koska tuotteen ei havaittu Tekesin ympäristössä tuovan riittävästi lisähyötyä.

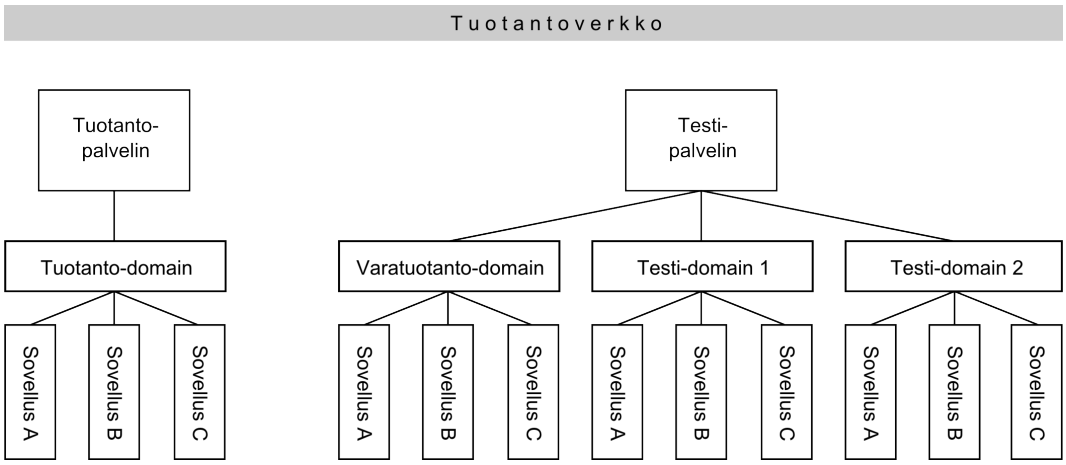
VMware ESX:ssä, kuten kaikissa natiivin virtualisoinnin ympäristöissä, voidaan ajaa lähes kaikkia kyseiselle prosessoriarkkitehtuurille sopivia käyttöjärjestelmiä. VMwaren aputoiminnot, kuten varmuuskopiointi, sekä osa virtualisointiominaisuuksista, kuten Enhanced VMXnet-verkkokortti sekä paravirtualisointi, toimivat silti vain määrättyjen käyttöjärjestelmien kanssa. Parhaiten tuettuja ovat Windowsin eri versiot sekä RedHat Enterprise Linux. Paravirtualisointi ei kuitenkaan toimi 64-bittisen Linuxin kanssa.

5.6 Sovelluskehitysympäristö virtuaalisena

Sovelluskehitysympäristön hallinta muuttui virtualisoinnin myötä lähes täysin. Vanhassa ympäristössä (kuva 19) realisoituivat monet kappaleessa 2 kuvatut ongelmat. Suurimpia näistä olivat:

- Yhden palvelinkoneen resurssit eivät riittäneet usean WebLogic-toimialueen ajamiseen
- Käytettiin samoja testiympäristöjä eri tarkoituksiin, minkä vuoksi oli epäselvää, kuka käyttää milloinkin mitä ympäristöä
- Ympäristöjä ei siivottu asennusten välillä, koska siivoaminen oli hankalaa
- Uuden ympäristön luominen oli vaivalloista, joten uusia ympäristöjä ei juuri koskaan luotu
- Vikatilanteissa ja päivitysten aikaan kaikki ympäristöt olivat poissa käytöstä
- Sovellukset piti kääntää aina tiettyä ympäristöä vastaan, mikä lisäsi turhaa manuaalista työtä
- Testiympäristöistä olisi pystynyt kohtuullisen vaivattomasti sotkemaan esim. tuotantotietokannan dataa, koska verkkoja ei oltu erotettu toisistaan

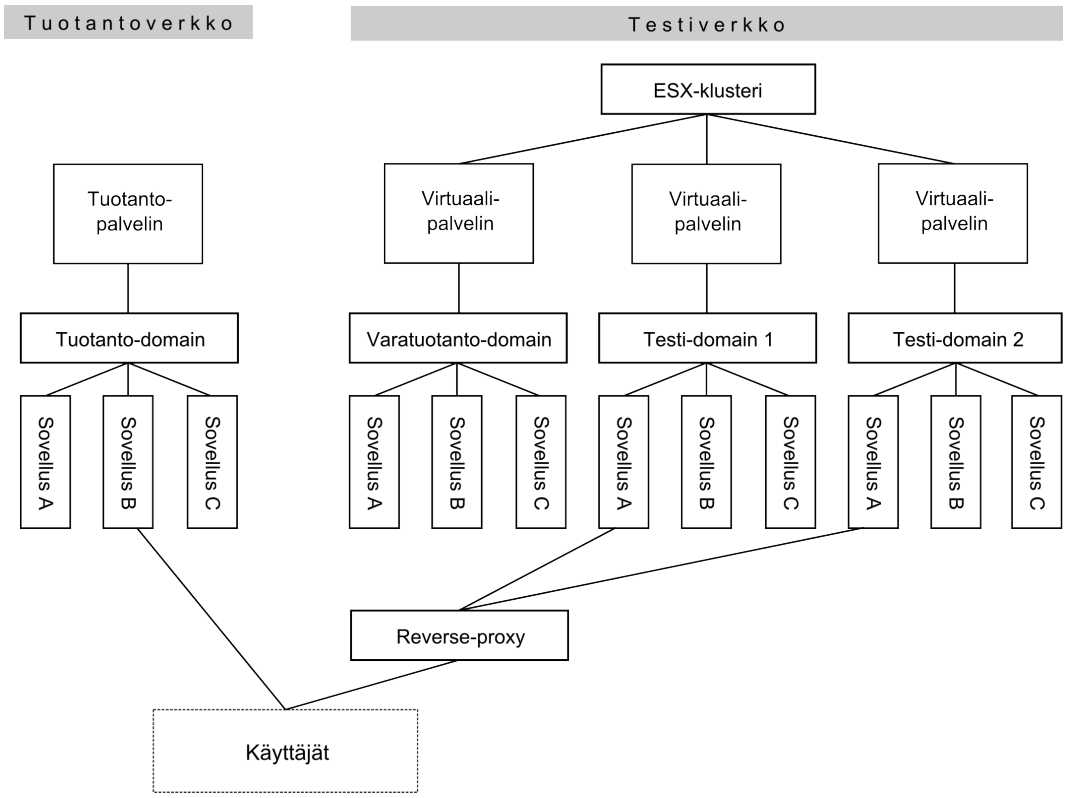
Suurin osa ongelmista ratkesi virtuaaliympäristössä ottamalla käyttöön yksi domain yhtä palvelinta kohden -periaate, joka ilmenee virtualisoinnin jälkeisestä kaavakuvasta 20.



Kuva 19: Sovellusympäristö ennen virtualisointia

Koska palvelimia pystyy luomaan helposti kloonaamalla, uuden ympäristön luominen muuttuu helpoksi, ja samalla eri tarkoituksiin voidaan luoda selkeästi omat palvelimet. Siivoamisesta ei tarvitse välittää, koska käytön jälkeen testiympäristön voi yksinkertaisesti tuhota kokonaan. Resurssiongelma siirtyy myös alemmalle tasolle – nyt tarvitsee vain pitää huoli, että virtuaaliklusterissa on tarpeeksi vapaata kapasiteettia. Kapasiteetin lisäys onnistuu esimerkiksi lisäämällä klusteriin uusia korttipalvelimia.

Virtualisoinnin yhteydessä ympäristö jaettiin verkkotasolla testi- ja tuotantoverkkoon. Testiverkosta sallitaan pääsy tuotantoon vain joihinkin pakollisiin palveluihin. Testiympäristöjen käyttöä helpottamaan väliin asennetaan nk. käänteisproxy (*reverse-proxy*), jonka taakse voidaan käyttäjiltä ikään kuin piilottaa eri palvelimilla toimivat palvelut. Ilman reverse-proxyä käyttäjien pitäisi aina tietää, millä palvelimella, missä portissa ja missä polussa kukin sovellus toimii. Reverse-proxyllä voidaan ohjata pyynnöt halutuille palvelimille: esim. http://proxy/applicationA-alpha/ ohjataan osoitteeseen http://10.1.2.3/app/applicationA/ ja http://proxy/applicationA-beta/ osoitteeseen http://10.1.2.4/app/applicationA/. Pääsivulle http://proxy/ asennetaan automaattisesti päivittyvä lista saatavilla olevista sovelluksista. Virtuaaliympäristössä tästä on tavallista enemmän hyötyä, koska testikoneita perustetaan ja tuhotaan usein. Reverse-proxy on käytännössä Linux-palvelin, jossa käytetään Apachea mod_proxy-moduulilla [www-43].



Kuva 20: Sovellusympäristö virtualisoinnin jälkeen

5.7 Suorituskyvyn testaus

Uuden ja vanhan ympäristön suorituskykyä on paras verrata mittaamalla sovelluksen nopeutta. Myös sovelluksen kääntämisessä ja tiedostojen siirrossa voi olla nopeuseroja, mutta näitä suoritetaan harvemmin, joten pieni ailahtelu suorituskyvyssä ei ole merkityksellistä. Ajetaan kuitenkin prosessoritehon ja levy-IO:n nopeustestit mahdollisten suurempien ongelmien löytämiseksi.

Testiin otetaan taulukon 2 mukaiset kokoonpanot.

Taulukko 2: Prosessorin suorityskykytestin kokoonpanot

	Virtuaalialusta	*Käyttöjärjestelmä*
Vanha kehitys-/tuotantoympäristö	-	RHEL 4 x86-32
Uusi tuotantoympäristö	-	RHEL 5.3 x86-64
Uusi kehitysympäristö	VMware ESX 3.5	RHEL 5.3 x86-64
Työasema	-	Windows 2008 x86-64
Virtualisoitu työasema	Sun xVM 2.1.4	CentOS 5.2 x86-32

	Prosessori	*Muisti*	*Levyjärjestelmä*
Vanha kehitys-/tuotantoympäristö	Intel Xeon 3.4 GHz	6 GB	HP SmartArray 6i
Uusi tuotantoympäristö	Intel Xeon E5320 1.8 GHz	16 GB	HP EVA (iSCSI)
Uusi kehitysympäristö	Intel Xeon E5320 1.8 GHz	6 GB	virtualisoitu
Työasema	Intel Core2 Q9550 3.4 GHz	8 GB	Intel ICH10R
Virtualisoitu työasema	Intel Core2 Q9550 3.4 GHz	1 GB	virtualisoitu

5.7.1 Sovellustesti

Käytetään sovellusten nopeuden mittaamiseen Jmeter-työkalua[www-44], joka lähettää palvelimelle HTTP-kutsuja imitoiden oikean käyttäjän WWW-selainta. Koska tarkastellaan vain testi- ja kehitysympäristön suorituskykyä, voidaan Jmeter määritellä avaamaan sovelluksen yleisimmin käytettyjä toimintoja, eikä tarvitse esimerkiksi tuotantoympäristön käytön pohjalta rakentaa profiilia, joka vastaisi tarkasti tuotantokäyttöä. Sovellustesti mittaa suurimmaksi osaksi prosessorin suorituskykyä, mutta myös muistin määrä, levyjärjestelmä ja verkko vaikuttavat tuloksiin.

Tyypillistä testiympäristön käyttöä vastaavasti suoritetaan testi yhdellä, viidellä ja kahdellakymmenellä yhtäaikaisella säikeellä. Sovelluspalvelin käynnistetään uudelleen ennen testin aloittamista välimuistien ja sessioiden tyhjentämiseksi. Tietokannan vaikutus eliminoidaan käyttämällä jokaisessa testissä samaa tietokantapalvelinta. Kyseessä on tehokas tuotantoverkon palvelin, jonka toiminta on testattu paljon suuremmilla kuormituksilla.

Jokainen säie suorittaa seuraavan loopin ilman keinotekoisia viiveitä, eli niin, että seuraava kutsu lähetetään välittömästi, kun edellisen vastaus on palannut:

```
For i = 1 to 20:
   Kirjaudu sovellukseen
   Sovelluksen navigaatiopalkkia käyttäen avaa projektinhakusivu
   For j = 1 to 5:
      Tee projektihaku ennalta määrätyllä organisaationimellä
      Tee laaja projektihaku vuosilta 1990--2005
      Tee projektihaku ilman ehtoja
```

```
      Avaa satunnainen projekti
      Hae projektin perustiedot
      Avaa projektiin liitetty asiakasorganisaatio
   Next j
   Kirjaudu ulos
Next i
```

Testi ei simuloi oikeaa käyttöä, eikä sen avulla voida määrittää esimerkiksi arviota maksimikäyttäjämäärästä. Testillä voidaan kuitenkin suoraan verrata eri ympäristöjen suorituskykyjä yksinkertaisesti vertaamalla testin läpivientiin kuluvaa aikaa. Testin aikana voidaan ympäristön laitteistoresursseja tarkkailemalla määrittää suorituskykyä rajoittava pullonkaula.

Koska sovellus käyttää sisäisesti useita säikeitä, voi prosessorien tai prosessoriydinten määrä vaikuttaa suuresti suorituskykyyn. Tämä kuitenkin tulee hyvin esille vasta sovelluksen palvellessa monia käyttäjiä yhtä aikaa, mitä mallintavat viiden ja kahdenkymmenen yhtäaikaisen testin tapaukset. Jmeteriä ajava työasema ei välttämättä selviä 20 yhtäaikakaisesta testisäikeestä, jolloin otetaan avuksi toinen ja mahdollisesti kolmas työasema.

5.7.2 Prosessoriteho

Mitataan virtualisoinnin tuoma lisäkuorma raakaa laskentatehoa vaativassa sovelluksessa ajamalla sama sovellus ilman virtualisointia ja virtualisoinnin kanssa muutamassa eri ympäristössä. Käytetään sovelluksena eri prosessoriarkkitehtuureille ja käyttöjärjestelmille sopivaa pi_css5-ohjelmaa [www-45], joka laskee halutun määrän piin desimaaleja. Tässä kahdeksan miljoonaa desimaalia tuottaa noin minuutin mittaisia testiajoja, joita ajetaan viisi jokaista ympäristöä kohden, ja näistä otetaan mediaani. pi_css5 on yksisäikeinen ohjelma, joka mittaa vain yhden prosessoriytimen suorituskykyä.

5.7.3 Levy-IO

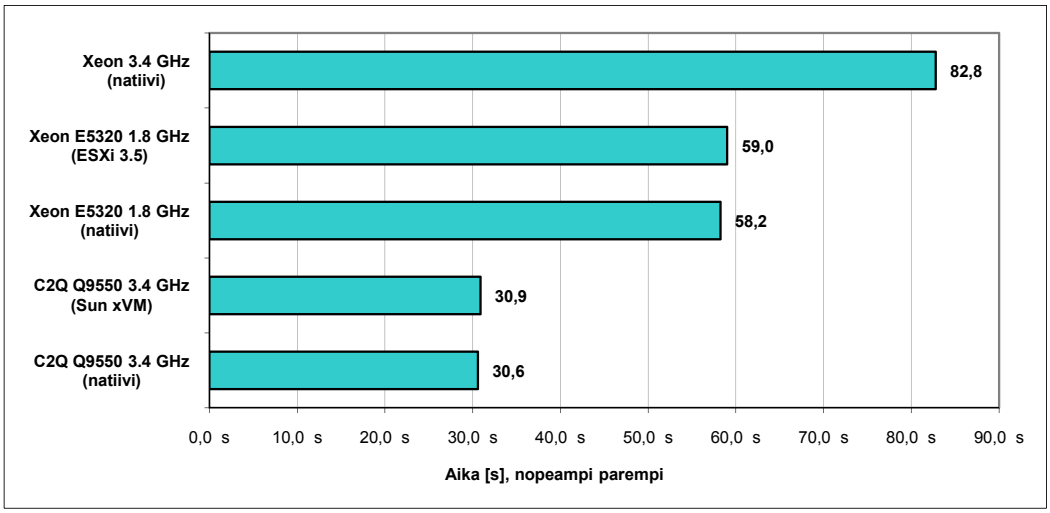
Mitataan levykirjoitusten nopeutta GNU-perustyökaluihin kuuluvalla dd-ohjelmalla [www-46]. Palvelinten muistin määrää vähennetään yhteen tai kahteen gigatavuun testien ajaksi, jotta testitiedosto ei mahtuisi kokonaan välimuistiin. Käytetään vähintään neljä kertaa keskusmuistin määrän kokoisia testitiedostoja. Esimerkiksi seuraava komento kirjoittaa 8 GiB testitiedoston:

```
date; dd if=/dev/zero of=testitiedosto bs=4M count=2048; sync; date
```

Kirjoitusnopeus saadaan jakamalla tiedoston koko date-komentojen ilmoittamien aikojen erotuksella. Sync-komento kirjoittaa välimuistissa olevat tiedostot levylle.

Testi testaa levyjärjestelmän ja levyjen nopeutta, sekä virtuaaliympäristössä levyn virtualisointikerroksen tehokkuutta. Vanha tuotanto- ja kehitysympäristö käyttää kahden 73 GB SCSI-levyn paikallista RAID1-järjestelmää, uusi tuotanto- ja virtuaaliympäristö erillistä HP:n SAN-järjestelmää iSCSI:lla ja työasema emolevylle integroidulla RAID-ohjaimella toteutettua viiden 750 GB SATA-levyn RAID5-pakkaa. Virtuaalikoneissa käytetään kokonaan valmiiksi allokoituja levykuvia, ei sparse-tyyppisiä levyjä.

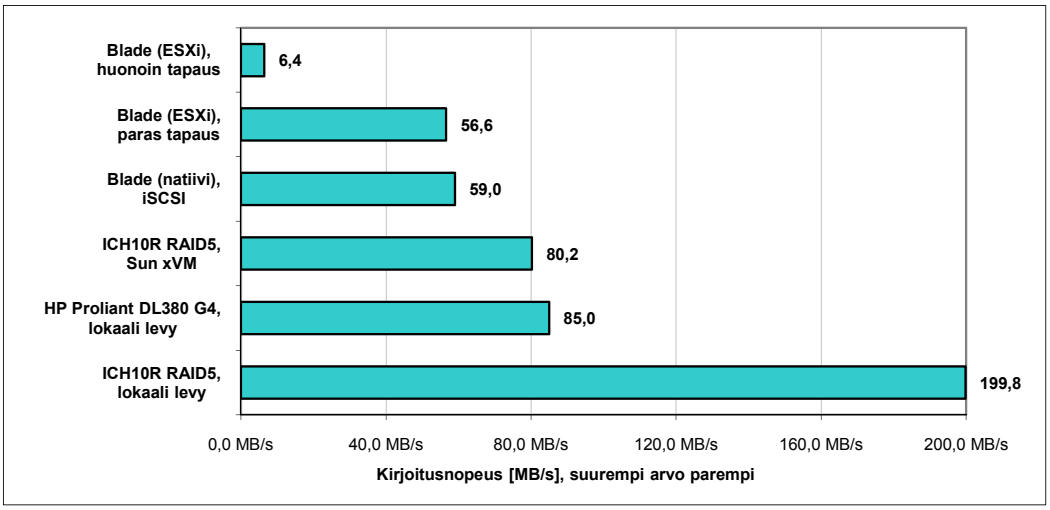
5.7.4 Testitulokset



Kuva 21: Prosessoritehon mittaustulokset

Prosessorin laskentatehotesti (kuva 21) osoittaa, että virtualisoinnin tuoma lisäkuorma puhtaassa laskennassa on mitätön, vain noin yhden prosentin luokkaa. Prosessorit asettuivat myös odotettuun järjestykseen: Vanhaa 90 nm NetBurst-tekniikkaa edustava Xeon 3.4 GHz jäi korkeasta kellotaajuudestaan huolimatta selvästi jälkeen modernimmasta 45 nm Core-tekniikkaa käyttävästästä Xeonista. Samaa tekniikkaa käyttävä Core2quad selvisi testistä lähes kaksinkertaisen kellotaajuutensa ansiosta noin puolessa ajassa Xeon E5320:een verrattuna.

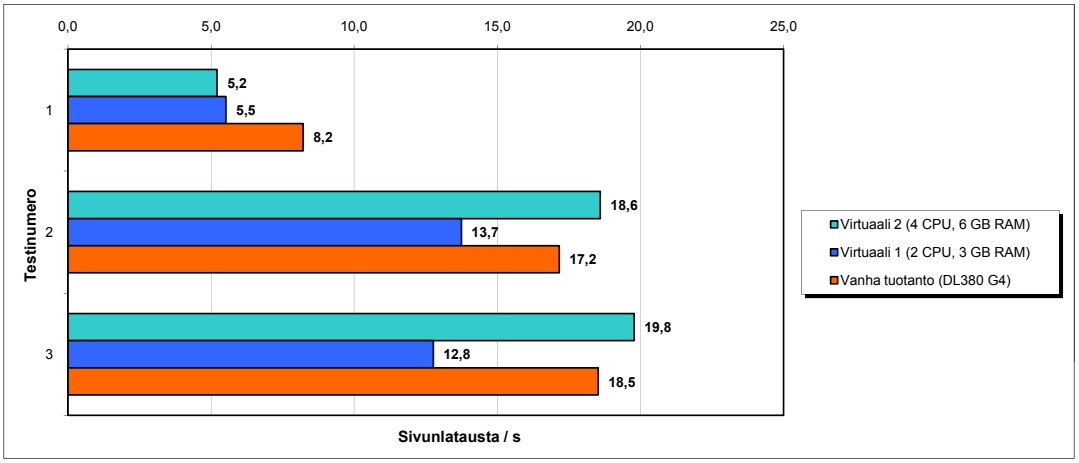
Levynopeustesti (kuva 22) todensi epäilyt virtuaaliympäristön heikosta levysuorituskyvystä. Virtuaaliympäristöille ominaiseen tapaan suorituskyky vaihteli paljon, mutta mittaukset osoittivat kirjoitusnopeuden tippuvan ajoittain käyttökelvottoman hitaaksi. Myös työaseman käyttämä Sunin virtuaaliympäristö leikkasi levynopeuden alle puoleen virtualisoimattomasta – nopeus tosin oli tämänkin jälkeen vielä parempi kuin ESX-virtuaaliympäristössä parhaimmillaan. Testissä ei mitattu levyjärjestelmien suorituskykyä monen käyttäjän raskaalla kuormalla, mutta silti työaseman alle 500 euron levyjärjestelmän ylivoimainen suorituskyky herättää epäilyjä, toimiiko blade-palvelimien käyt-



Kuva 22: Levynopeuden mittaustulokset

tämä SAN-järjestelmä varmasti oikein. Moderni yksittäinen työaseman SATA-kiintolevy kirjoittaa noin 100 MB/s [48], joten palvelimen odottaisi pystyvän vähintään samaan.

Levyvälimuisti tasoittaisi suorituskykyeroja jonkin verran, mutta sovelluspalvelimilla muisti kuluu yleensä kokonaan sovelluksille, jolloin välimuistille jää hyvin vähän toimintatilaa. Virtuaaliympäristön levyjen hitaudesta on saatu myös suoraa palautetta ympäristöä testanneilta sovelluskehittäjiltä, joten kyseessä ei ole vain mittauksissa esiin nouseva ongelma.



Kuva 23: Rahoituksenhallintasovelluksen nopeusmittauksen tulokset

Rahoituksenhallintasovelluksen testissä 1 järjestelmää käyttää yksi “testikäyttäjä” eli Jmeterin säie, joka suorittaa luvussa 5.7.1 kuvattua testiä. Testissä 2 yhtäaikaisia säikeitä on 5 ja kolmannessa testissä 20. Jmeteriä ajavan työaseman prosessorikäyttö 20 säikeen testissä oli noin 80 %, joten testin suorittaminen ei vaatinyt useamman työaseman klusterointia.

Prosessorinopeustestistä poiketen vanha tuotantopalvelin oli yhdellä säikeellä selvästi nopeampi kuin virtuaaliympäristöt (kuva 23). Muistin loppuessa virtuaalipalvelimen suorituskyky olisi voinut romahtaa, mutta tästä ei ollut kyse, koska 6 GB:n muistilla varustettu virtuaalipalvelin antoi samat tulokset kuin 3 GB:lla. Prosessorikuorma oli noin 30 % kahdesta ytimestä tai 15 % neljästä. Levyaktiviteetti rajoittui lähinnä lokitiedostojen kirjoittamiseen.

Jäljelle jäävä muuttuja on verkko, jota sovellus käyttääkin intensiivisesti. Vähänkin kasvanut verkkolatenssi näkyy nopeasti suorituskyvyssä: jo puoli millisekuntia kasvanut viive pakettia kohden aiheuttaa helposti 50 ms eron sivua kohden. Sivulatauksia testissä 1 on 1380 kpl, mikä tarkoittaa testille $0.050\ \mathrm{s} \cdot 1380 = 69\ \mathrm{s}$ pidempää kokonaiskestoa. Jmeterin mittaamat keskimääräiset sivunlatausajat olivat 119 ms vanhalle tuotantopalvelimelle ja 180 ms (2 vCPU) sekä 190 ms (4 vCPU) virtuaaliympäristölle. Vanha tuotantoympäristö oli tässä testissä sivujenlatausmäärällä mitaten jopa 57 % tehokkaampi kuin virtuaaliympäristö.

Viiden ja kahdenkymmenen säikeen testeissä vanhan ympäristön heikompi CPU-teho muodostui pullonkaulaksi, jolloin virtuaalikoneet pärjäsivät paremmin hitaammasta verkosta huolimatta. Viidellä säikeellä vanha tuotanto oli edelleen 25 % tehokkaampi kuin kahden virtuaaliprosessorin ympäristö, mutta hävisi neljän vCPU:n ympäristölle 7,5 %. Ympäristöjen keskimääräiset CPU-kuormat olivat vastaavasti 86 % ja 63 %. Kahdenkymmenen säikeen testissä kevyemmän virtuaaliympäristön muisti kävi vähiin aiheuttaen Java VM:ssä tiheää roskankeruuta (*garbage collection*) ja näin kuluttaen ylimääräistä CPU-aikaa.

6 Saavutettujen etujen ja riskien analysointi

Tässä luvussa esitetään muutosehdotuksia organisaation sovelluskehitysympäristöön. Uudet toimintatavat vaativat joitakin ohjelmisto- ja laitehankintoja, ja uudet ohjelmistot sekä laitteet vaativat muutoksia prosesseihin ja käytäntöihin.

6.1 Uudet toimintatavat

Virtualisoitu sovelluskehitysympäristö tukee vahvasti ketterää kehitystä. Kehitysprojektin uuden iteraatiokierroksen aloittaminen realisoituu uuden kehitysympäristön perustamisella, mikä tapahtuu tehokkaasti palvelinpohjasta (*template*), joka on käytännössä tuotantojärjestelmän asetuksilla toimiva virtuaalikone. Eri komponenttien kehitystä varten voidaan luoda useita ympäristöjä, jotka yhdistetään hyväksymistestausta ja tuotantoon vientiä varten. Aikaa säästetään selvästi, kun ympäristöjen konfigurointitarve on minimaalinen ja vanhoja testiympäristöjä ei tarvitse siivota uusien tieltä.

Lisäksi ympäristöjä on aina käytettävissä riittävä määrä, joten tietyn komponentin kehittämisen ei tarvitse odottaa kehitys- tai testausympäristön vapautumista toisesta käyttötarkoituksesta. Rajoitteena on tietenkin käytettävien palvelinresurssien sekä hallinnointityön määrä – kolme ympäristöä sovellusta kohden on järkevä käytännön yläraja.

6.2 Lisenssiongelmat

Kustannussäästöt ovat useimmiten yksi tärkeimpiä virtualisoinnin lähtökohtia, mutta kasvaneen palvelinmäärän myötä ohjelmistolisensseistä voi koitua yllättävän suuria lisäkustannuksia. Monien palvelinohjelmistojen lisensointiehdot virtuaaliympäristöjä varten ovat epäselviä ja hintoja on mahdoton selvittää ilman yhteydenottoa myyjään. Ellei muuta mainita, tarvitaan yhtä virtuaalipalvelinta varten täsmälleen samanlainen lisenssi kuin fyysisellekin palvelimelle.

Tekes käyttää lähes yksinomaan kaupallisia ohjelmistoja niin käyttöjärjestelmissä, sovelluspalvelimissa kuin tietokannoissakin. Esimerkiksi WebLogic- ja Oracle-lisenssit muodostavat erittäin merkittävän kustannuserän jo vanhassakin ympäristössä. RedHat tarjoaa virtuaaliympäristöön jonkin verran halvempaa lisenssiä – tosin täysihintaisenkin lisenssi on edellä mainittuihin *enterprise*-tuotteisiin verrattuna halpa.

Ongelmia syntyy, kun virtualisoinnin myötä kahden WebLogic-palvelimen asemesta näitä onkin lähes kymmenen, ja määrä vaihtelee usein. Viisi käyttäjää sallivaa *developer*-versiota voidaan käyttää joissakin testiympäristöissä, mutta esimerkiksi integraatiotesteihin ja hyväksymistestaukseen se ei riitä.

Lisenssikustannuksien pienentämiseen yksinkertaisin ratkaisu on käyttää halvempia tai ilmaisia tuotteita. Ominaisuuksien ja suorituskyvyn puolesta Oracle-tietokannat voisi korvata hyvin MySQL-kannoilla; myös jälkimmäiselle saa halutessaan ostettua kaupallisen tukipalvelun. WebLogicin korvaaminen toisella sovelluspalvelintuotteella nykyisten sovellusten osalta voi olla liian suuri projekti ollakseen nettonykyarvoltaan kannattava, mutta uusia sovelluksia kehitettäessä ne olisi hyvä alusta alkaen rakentaa toimimaan esimerkiksi JBossilla tai muulla ilmaisella alustalla. RedHat Enterprise Linuxin voisi korvata testiympäristöissä aivan suoraan ja myös monilla tuotantoympäristön palvelimilla hyvin helposti CentOS-käyttöjärjestelmällä tai jopa Ubuntulla. Täytyy kuitenkin pitää huoli, että esimerkiksi Oracle ja WebLogic asennetaan sellaiselle käyttöjärjestelmälle, joka on virallisesti tuettu – niin kauan kuin näitä vielä käytetään.

6.3 Suosituksia *use case* -organisaation ympäristön jatkokehitykseen

Organisaation verkon käytöstä pitäisi olla selkeä suunnitelma, jotta päästäisiin tietoturvalliseen, mutta silti helppokäyttöiseen kokonaisuuteen. Testi- ja tuotantoverkko tulee erottaa toisistaan palomuurilla niin, ettei testiympäristöistä ole mahdollista päästä lainkaan tuotantojärjestelmiin. Tiukka palomuuri edellyttää samalla kaikkien tarvittavien palveluiden tuomista testiverkon puolelle, jolloin päästään eroon nykyisestä toimintamallista, jossa sekä testi- että tuotanto käyttävät mm. samoja tunnistus-, hakemisto- ja dokumenttipalvelua. Palveluiden tuominen testiverkkoon tehostaa testausta ja vähentää riskejä.

Verkkouudistus mahdollistaisi myös vapaammat etäyhteydet sovellustoimittajille. Tekesillä on pyritty siihen, että sovelluskehittäjien ei tarvitsisi tehdä kehitystä organisaation tiloissa, vaan he voisivat tehdä kaiken integraatiotestausta lukuun ottamatta omissa toimipisteissään. Etäyhteyksillä testaus, jopa integraatiotestaus, voitaisiin suorittaa turvallisesti etäyhteyden yli, mikäli sovellustoimittajille voitaisiin avata vapaa pääsy testiverkossa oleviin palveluihin. Organisaatio voisi tarjota testiverkkoon esimerkiksi TLS-salattuja OpenVPN-etäyhteyksiä. Tämä toisi suoraan kustannussäästöjä, mikäli nykyisestä kaupallisesta VPN-tuotteesta luovuttaisiin.

Testiverkon palvelimien varmuuskopiointiin virtuaalialustojen tarjoamien automatisoitujen ratkaisuen, esim. VMware Consolidated Backupin käyttäminen toisi selkeyttä ja vähentäisi manuaalista työtä. Vaikka testipalvelimilla ei olekaan liiketoiminnan kannalta tärkeää dataa, itse ympäristöjen asentamiseen on käytetty ja käytetään paljon työtunteja. Testiympäristössä lisäksi vahinkoja sattuu huomattavasti enemmän kuin tuotannossa, joten varmuuskopioille olisi myös useammin tarvetta. Jossain määrin varmuuskopiointia voidaan korvata tilannekuvilla (*snapshot*) esimerkiksi ennen käyttöjärjestelmäpäivityksiä, mutta tilannekuvat hidastavat virtuaalikoneiden toimintaa ja niiden ottaminen on manuaalista työtä, joka helposti unohtuu tai jätetään kiireessä tekemättä.

Virtuaali-isäntäpalvelimia seuraavan kerran uusittaessa kannattaa sekä testi- että tuotantoverkkoon ottaa täsmälleen samaa prosessorimallia, jolloin virtuaalikoneiden siirtäminen näiden välillä onnistuu entistäkin paremmin. Nykyisessä järjestelmässä ajosta jäädytettyjen tilannekuvien siirtäminen ei onnistu, koska toinen ympäristö käyttää AMD:n ja toinen Intelin prosessoreja.

6.3.1 Ratkaisuja suorituskykyongelmiin

Suorituskykytesteissä esiin nousseeseen levy-IO:n hitauteen on monia ratkaisuja. Ensimmäiseksi kannattaa tutkia, mitkä virtuaalikoneet käyttävät eniten levyä. VMware Infrastructure Clientissä asian voi tarkistaa suoraan suorituskykygraafista. Tarkempaa statistiikkaa saa Linuxissa esim. iostat-ohjelmalla tai monitorointityökaluilla. Ahkerasti swap-muistia käyttävä virtuaalikone voi niin ikään toimia pullonkaulana muiden koneiden levy-IO:lle, joten samalla kannattaa tarkistaa koneiden muistin käyttö.

Levytilaa enemmän kuluttava, mutta nopeampi tapa virtuaalilevyjen käsittelemiseen on käyttää kokonaan esiallokoituja (*flat file*) tiedostoja haja-allokoitujen (*sparse*) asemesta. Jos koneiden kloonaamiseen käytetään VMwaren vmkfstools-ohjelmaa, luodaan esiallokoitudut tiedostot valitsemella `-d thick` ja haja-allokoidut valitsimella `-d thin`. Joka tapauksessa virtuaalilevyyn kohdistuvat operaatiot joutuvat kulkemaan isäntäpalvelimen tiedostojärjestelmän läpi, mikä hidastaa toimintoja jonkin verran. Esimerkiksi Xeniä ja LVM-pohjaisia levyjä käytettäessä tällaista ongelmaa ei ole, koska virtuaalikoneelle voidaan osoittaa suoraan haluttu laite (*raw device*).

Kolmas mahdollisuus on iSCSI-kohteen osoittaminen virtuaalikoneelle suoraan SAN-järjestelmästä ohi virtuaalijärjestelmän. Tähän liittyy kuitenkin monia käytännön ongelmia. Hankalin näistä lienee verkkokonfiguraatio: SAN toimii yleensä eristetyssä verkossa, jota varten virtuaali-isäntäkoneella pitäisi olla erillinen verkkosovitin. Lisäksi kyseinen verkko pitäisi saada konfiguroitua sekä isännälle että sen kautta itse virtuaalikoneelle. Suuri verkkoliikenne voi puolestaan tuottaa erilaisia ongelmia virtuaalikoneille, varsinkin, mikäli käytössä ei ole optimoitua, paravirtualisoitua verkkosovitinta. iSCSI:n käyttäminen myös jossain määrin mitätöi virtuaaliympärisön hyötyjä: mm. kloonausta on vaikea hyödyntää, jos palvelimen tiedot eivät ole kloonautuvalla medialla.

Sovelluspalvelimien suorituskyky on havaittu tällä hetkellä riittäväksi, mutta tulevaisuudessa asia voi muuttua nopeastikin. Sovelluspalvelimien klusterointi tarjoaisi skaalautuvuutta helposti palvelinkoneita lisäämällä (ks. 6.3.5).

6.3.2 Ratkaisuja levytilaongelmiin

Virtuaaliympäristön levytilantarve on ollut jonkin verran suunniteltua suurempi. Virtualisoidut tiedosto- ja tietokantapalvelimet tarvitsevat monesti yli 100 GB levytilaa

palvelinta kohden, ja levytilantarvetta on virtuaalikoneita luotaessa ennemminkin ylikuin aliarvioitu, jolloin käyttämätöntä, mutta varattua tilaa on paljon. Koneista on myös varmuuskopiointimielessä otettu klooneja, jotka ovat jääneet viemään levytilaa.

Suorituskykyä heikentävä, mutta levytilaa erittäin paljon säästävä vaihtoehto on käyttää haja-allokoituja (*sparse*) virtuaalilevyjä. Tämä helpottaa tarvittavan levykoon arvioimista, koska tilantarve voidaan huoletta yliarvioida ilman, että isäntäkoneen levyä varattaisiin heti arvioidun määrän verran. Organisaation Linux-sovelluspalvelimet tarvitsevat tyypillisesti vain noin 10 GiB levyä, mutta levytilan loppumisen välttämiseksi virtuaalikonetta luotaessa on sille tehty 30 GiB virtuaalilevy. Mikäli tilantarve pysyy 10 GiB:ssä, pidetään jatkuvasti 20 GiB levyä turhaan varattuna. Samaan levytilaan saadaan ainakin kaksinkertainen määrä virtuaalikoneita käyttämällä sparse-levyjä.

Yhteistä dataa voidaan säilyttää virtuaalikoneiden paikallisten levyjen asemesta myös verkkojaoissa. Esimerkiksi debug-parametreilla ajattevan sovelluksen lokitiedostot voivat olla gigatavujen kokoisia, mutta näitä lokeja ei ole hyödyllistä säilöä pitkältä ajalta virtuaalikoneen paikalliselle levylle. Kompressoituja lokeja voitaisiin säilyttää keskitetyssä paikassa esim. kahden viikon kiertoajalla.

6.3.3 Monitorointi ja statistiikan keruu

Organisaatiolla on jo käytössä tai testikäytössä muutamaa eri monitorointituotetta. Käyttöä tulisi kuitenkin laajentaa ja pyrkiä keskittämään kaikki monitorointidata yhteen paikkaan, josta se olisi helposti seurattavissa.

Sovelluskehitystä ja ympäristön jatkokehitystä helpottaisi paljon statistiikka sovellusten käytöstä. Esimerkiksi yhtä aikaa sisään kirjautuneiden käyttäjien määrä, sivupyyntöjen määrä sekunnissa ja sivujen latausajat antaisivat kehittäjille arvokasta tietoa sovelluksen suorituskyvystä, mahdollisista pullonkauloista sekä mahdollisesti riittämättömistä laiteresursseista. Tilastojen keruu voitaisiin toteuttaa joko erillisellä statistiikkapalvelulla, johon sovellus itse lähettää jatkuvasti dataa, tai vaihtoehtoisesti ulkoisesti analysoimalla sovelluksen lokitiedostoa. Ensin mainittu olisi joustavampi ja tyylikkäämpi ratkaisu, mutta vaativampi toteuttaa – sitä varten pitäisi määritellä uusi rajapinta, joka otettaisiin käyttöön eri toimittajien sovelluksissa.

6.3.4 Reverse-proxy

Sovelluksille menevät pyynnöt kulkevat reverse-proxy-palvelun läpi käytön ja konfiguroinnin helpottamiseksi. Tekesin ympäristössä reverse-proxy-palvelimia tarvitaan peräti kolme: DMZ-alueelle julkisesta Internetistä tulevia pyyntöjä varten, tuotantoympäristön sisäverkkoon Tekesin työntekijöitä varten sekä testiverkkoon testausta ja sovelluskehitystä varten (kuva, Liite A).

Proxyn läpi näkyvä URL	Alkuperäinen URL	Kommentti
http://proxy/A	http://tuotantopalvelin1:7400/A/app	Sovelluksen A tuotantoversio
http://proxy/B	http://tuotantopalvelin1:7300/B	Sovelluksen B tuotantoversio
http://testiproxy/A12/A	http://a12test:7400/A/app	A:n kehitysversio v1.2
http://testiproxy/A12/B	http://a12test:7300/B	B sovelluksen A v1.2:n kehitysympäristössä
http://testiproxy/prod/A	http://a11test:7400/A/app	A:n tuotantoversion (v1.1) testaus

Taulukko 3: Esimerkki reverse-proxy-poluista

Hyvä nimeämiskäytäntö testiympäristöjä varten on kategorisoida polut ympäristöjen mukaan (taulukko 3). Toinen hyödyllinen käytäntö on sijoittaa proxy-palvelimen juurikontekstipolkuun automaattisesti päivittyvä lista proxyn ohjaamista palveluista.

6.3.5 WebLogic-palvelimien klusterointi

Tuotannon sovelluspalvelimien klusteroinnilla saavutettaisiin yhdellä kertaa nopeampi vasteaika, parempi saatavuus sekä helpotusta ylläpitoon. Tehokkuus paranee, kun kuormantasaimella jaetaan pyynnöt tasaisesti molemmille palvelimille. Vikatilanteessa kuormantasain ohjaa kaikki pyynnöt toimivalle palvelimelle, jolloin menetetään vain vikaantuneella palvelimella olleet sessiot. Palvelimet käyttävät kuitenkin yhteistä tietokantaa, joten datan menetys kohdistuu vain juuri viikaantumishetkellä olleisiin tallentamattomiin tietoihin.

Ylläpito helpottuu, koska toinen palvelimista voidaan irroittaa klusterista päivityksiä varten ilman järjestelmän toimintakatkosta. Päivityksen jälkeen sama tehdään vastaavasti toiselle palvelimelle. Merkittävänä lisäetuna vältytään kokonaan erillisen varajärjestelmän ylläpidolta – olettaen, että tietokanta on myös klusteroitu tai muulla tavoin varmistettu.

Klusterointia testattaessa WebLogic-klusteri toimi pääasiassa hyvin: samat toiminnot pystyi helposti suorittamaan yhdellä kertaa molemmille klusterin koneille, eikä ongelmia esimerkiksi tietokantayhteyksien tai JMS-jonojen kanssa esiintynyt lainkaan. Jonkin verran ongelmia sen sijaan tuotti kuormantasaimena käytettäy Apachen Reverse-proxy-moduuli, joka ei kunnolla havainnut toisen koneen klusterin koneen putoamista vikatilanteessa.

6.4 Yhteenveto

Sovelluskehityksen tehostamiseksi ja tietoturvan sekä saatavuuden parantamiseksi Tekesin kannattaa toteuttaa seuraavia muutostoimenpiteitä:

- Verkon jakaminen tiukasti tuotanto- ja testiverkkoihin
- Varmuuskopioinnin toteuttaminen virtuaalikonetasolla käyttöjärjestelmä- tai sovellustason asemesta
- Levytilan säästäminen mitoittamalla virtuaalikoneiden levykoot konservatiivisemmin
- Levynopeuden parantaminen käyttämällä erillistä SAN- tai NAS-järjestelmää tuotantoympäristölle ja testiympäristölle
- Helpottamalla palvelinten osoitteiden hallintaa reverse-proxyllä
- Klusteroimalla sovelluspalvelimet vältytään varaympäristön ylläpidolta ja nostetaan saatavuutta

Kustannussäästöjä saadaan aikaan tehostamalla toimintaa edellä mainituilla toimenpiteillä sekä merkittävissä määrin myös lisenssisäästöillä, kun hintavia enterprise-tuotteita vaihdetaan vapaisiin ohjelmistoihin. Varsinkin uusia projekteja aloitettaessa on syytä harkita tarkkaan, tarjoaako kymmeniä tuhansia euroja maksava kaupallinen tuote, esimerkiksi tietokanta tai sovelluspalvelinohjelmisto, mitään lisäarvoa vastaaviin vapaan lähdekoodin ohjelmistoihin nähden.

7 Loppusanat

Työn tarkoituksena oli tutkia virtualisoinnin tuomia mahdollisuuksia sekä mahdollisia haittapuolia erityisesti sovelluskehityksen kannalta. Uusilla ratkaisuilla pyritään tehostamaan monimutkaisen palvelinympäristön toimintaa ja hallintaa sekä saavuttamaan kustannussäästöjä. Työhön kuului myös virtualisoidun ympäristön suunnittelu Tekesille sekä uuden ympäristön etujen ja haittojen analysointi.

Lähkökohtaisesti virtualisointi tarjoaa uuden abstraktiotason laitteiston ja ohjelmiston väliin. Tätä uutta tasoa hyödyntämällä voidaan palvelinympäristöä hallita täysin uusilla toimintatavoilla, jotka ovat aiemmin olleet teknisesti joko mahdottomia tai erittäin hankalasti toteutettavia. Uusilla toimintatavoilla pyritään parempaan tehokkuuteen ja kustannussäästöihin, mutta samalla vaaditaan enemmän asiantuntemusta ylläpitäjiltä ja virtuaaliympäristön käyttäjiltä.

Työssä kävi ilmi, että osa virtualisoinnin usein mainostetuista hyödyistä ovat todellisia ja oikeasti lisäarvoa tuovia, mutta myös haitat ovat konkreettisia. Hyödyllisimmiksi teknisiksi uudistuksiksi havaittiin *snapshotit*, uusien palvelimien perustamisen ja kofiguroinnin nopeus ja helppous sekä fyysisten palvelimien määrän vähentäminen resurssien konsolidoinnilla. Virtuaalisen konesalin helppokäyttöinen, visuaalinen hallinta tukee vahvasti edellä mainittuja hyötyjä.

Haittoina mainitaan usein ailahteleva tai heikentynyt suorituskyky sekä yhteensopivuusongelmat. Näistä suorituskykyongelmat todettiin todellisiksi, mutta yhteensopivuusongelmia ei enää nykyisillä virtualisointiratkaisuilla havaittu. Tästä huolimatta monet sovellusvalmistajat eivät edelleenkään tarjoa tukea, mikäli sovellusta käytetään virtuaaliympäristössä. Tämä on virtualisointiprojektia ratkaisevasti rajoittava tekijä.

Työn aikana Tekesillä suunniteltiin, evaluoitiin ja otettiin käyttöön virtualisoitu sovelluskehitysympäristö liiketoiminnan kannalta merkittävimmille sovelluksille. Vaikka ympäristö on jo täysimääräisesti toiminnassa, jatkokehitystä tarvitaan tehokkuuden ja tietoturvan parantamiseksi. Ohjelmistovaihdoksilla voidaan saavuttaa merkittäviä kustannussäästöjä.

Viitteet

[1] L. Lawson, "The Benefits of Using Virtualized Testing with SOA". IT Business Edge, [Online], 2007-08-20, haettu 2009-02-20.
http://www.itbusinessedge.com/cm/community/features/interviews/blog/the-benefits-of-using-virtualized-testing-with-soa/?cs=22617

[2] K. Niemistö, "Ympäristöystävällinen IT: VMware - Energian säästöä palvelinten virtualisoinnilla". VMware Finland, [Online], 2008-04-03, haettu 2009-02-14. (Sivu 4.)
http://www.netvideo.fi/ttl/20080403/4/4-Niemisto.pdf

[3] I. Singh, B. Stearns, M. Johnson ja Sun Microsystems Enterprise Team, *Designing Enterprise Applications with the J2EE Platform,* 2. painos. Addison-Wesley, 2002, ISBN 0-201-78790-3. (Luku 4.3.)

[4] J. Petersen, "Benefits of using the n-tiered approach for web applications". Adobe, inc, [Online], 2009, haettu 2009-03-02.
http://www.adobe.com/devnet/coldfusion/articles/ntier.html

[5] Oracle / BEA, "Using WebLogic Server Clusters". [Online], 2007-03-30, haettu 2009-02-24. (Luvut 2 ja 5.)
http://download.oracle.com/docs/cd/E11035_01/wls100/pdf/cluster.pdf

[6] Oracle / BEA, "WebLogic Server 8.1 Admin guide". [Online] 2006-06-28, haettu 2009-02-24. (Luvut 1-13–1-20 ja 2.)
http://download.oracle.com/docs/cd/E13222_01/wls/docs81/pdf/adminguide.pdf

[7] R. A. Boggs, "The SDLC and Six Sigma". Florida Gulf Coast University, [Online], 2004-08-11, haettu 2009-02-16. (Sivu 38.)
http://www.iacis.org/iis/2004_iis/PDFfiles/Boggs.pdf

[8] L. Segal, "Key Considerations in Performance Testing". Testware associates, inc, [Online], 2002-05-29, haettu 2009-02-25. (Sivut 1 ja 3.)
http://www.testwareinc.com/resources/articles/Key_Considerations_in_Performance_Testing.pdf

[9] G. Popek ja R. Goldberg, *Formal requirements for virtualizable third generation architectures.* 1974-07.
DOI: 10.1145/361011.361073, ISSN: 0001-0782. (Sivut 413 ja 417.)

[10] T. Liston ja E. Skoudis, "On the Cutting Edge: Thwarting Virtual Machine Detection". IntelGuardians, Inc, [Online], 2006-07-21, haettu 2009-03-03.

(Sivu 4 ja 21–26.)
http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf

[11] G. R. Hook, ja R. Chitor, "Capped and Uncapped Partitions in IBM POWER5". IBM Solutions Enablement, Systems group, [Online], 2005-05, haettu 2009-03-05. (Sivut 1 ja 4.)
http://www-03.ibm.com/servers/enable/site/peducation/wp/40d6/40d6.pdf

[12] A. Kehrer, "IBM Targets Small Business with z10 Mainframe". Linux Magazine, [Online], 2008-10-25, haettu 2009-02-27.
http://www.linux-magazine.com/online/news/ibm_targets_small_business_with_z10_mainframe

[13] A. Tucker ja D. Comay, "Solaris Zones: Operating System Support for Server Consolidation". Sun Microsystems, Inc., [Online], 2004-03-07, haettu 2009-03-05. (Sivu 1.)
http://www.usenix.org/events/vm04/wips/tucker.pdf

[14] D. Zakrisson ja H. Pötzl, "Paper". Linux-vServer, [Online], 2008-08-10, haettu 2009-02-25:
http://linux-vserver.org/Paper

[15] J. Hurst, "Operating System Protection And Rings". Global Information Assurance Certification, [Online], 2007-02-06, haettu 2009-03-05. (Luvut "OS Rings" ja "Rings as Gated Communities".)
http://www.giac.org/resources/whitepaper/architecture/92.php

[16] A. Dornan, "Intel VT vs. AMD Pacifica". Network Computing, [Online], 2005-11-01, haettu 2009-03-10. (Luku "Drilling Down".)
http://www.networkcomputing.com/showArticle.jhtml?articleID=172302134

[17] Intel corp., "Intel Virtualization Technology (VT) in Converged Application Platforms". [Online], 2007-01-25, haettu 2009-03-10. (Sivut 8–10.)
http://www3.intel.com/design/intarch/papers/316337.pdf

[18] Channabasavaiah, Holley ja Tuggle, "Migrating to a service-oriented architecture", osa 1. IBM, [Online], 2003-12-23, haettu 2009-03-10.
http://www-128.ibm.com/developerworks/library/ws-migratesoa/

[19] J. N. Matthews, E. M. Dow, T. Deshane, W. Hu, J. Bonqio, P. F. Wilbur ja B. Johnson, *Running Xen: A hands-on guide to the art of virtualization.* Prentice Hall, 2008-04-16. ISBN 978-0132349666. (Sivut 6–7.)

[20] R. J. Creasy, “The Origin of the VM/370 Time-sharing System”. IBM, [Online], 1981-09, haettu 2009-03-09. (Sivu 1.)
http://www.research.ibm.com/journal/rd/255/ibmrd2505M.pdf

[21] Xen / Citrix, “Paravirtualization”. [Online], 2008, haettu 2009-02-02.
http://staging.xen.org/about/paravirtualization.html

[22] A. Kivity, “Paravirtualization is dead”. [Online], 2008-04-10, haettu 2009-02-02.
http://avikivity.blogspot.com/2008/04/paravirtualization-is-dead.html

[23] Sun Microsystems, “The Java HotSpot Performance Engine Architecture whitepaper”. Sun Developer Network, [Online], haettu 2009-03-04. (Luku 3.)
http://java.sun.com/products/hotspot/whitepaper.html

[24] VMware, “VMware VMotion: Live migration of virtual machines without service interruption”. [Online], 2009-10-19, haettu 2009-03-11.
http://www.vmware.com/pdf/vmotion_datasheet.pdf

[25] S. Whitehouse, “Linux kernel 2.6.19 changelog”. [Online], 2006-11-20, haettu 2009-03-11. (Osat merkinnällä “[GFS2]”).
http://www.kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.19

[26] Hewlett-Packard, “HP Integrity Virtual Machines A.03.00 Release Notes”. [Online], 2007-04-23, haettu 2009-03-12. (Sivu 14.)
http://www.docs.hp.com/en/T2767-90076/T2767-90076.pdf

[27] Inv Softworks LLC, ”NTFS Sparse Files For Programmers”. FlexHex Articles and Sources, [Online], 2007, haettu 2009-03-12. (Luku “Sparse Files – What Is It?”)
http://www.flexhex.com/docs/articles/sparse-files.phtml

[28] D. Davis, “What is a VMware snapshot*?*“. Petri IT Knowledgebase, [Online], 2009-01-08, haettu 2009-03-03. (Luvut 1–2.)
http://www.petri.co.il/virtual_vmware_snapshot.htm

[29] D. Sumsky, “VMware ESX server 3.x snapshots - introduction”. “Dsumsky lines ...”, [Online], 2008-06-02, haettu 2009-03-03.
http://dsumsky.blogspot.com/2008/06/vmware-esx-server-3x-snapshots.html

[30] Oracle, “Support Position for Oracle Products Running on VMWare Virtualized Environments”. Oracle Metalink, note 249212.1, [Online], 2007-11-16.

[31] N. Lewis, “Virtualization Security - Even the latest network technologies can introduce security holes”. IT Security, [Online], 2009-06-17, haettu

2009-03-08.
http://www.itsecurity.com/features/virtualization-security-061708/

[32] H. van Vliet, *Software Engineering: Principles and Practice.* Wiley, 2008. ISBN 978-0470031469. (Sivut 10–15 ja 47–51.)

[33] S. Elliott, "Agile Project Management: Seminar on Current Trends in Software Industry". University of Helsinki, Dept. of Computer Science, [Online], 2008-03-24, haettu 2009-03-08.
http://www.cs.helsinki.fi/u/paakki/Elliott.pdf

[34] J. R. Erenkrantz, "3rd Workshop on Open Source Software Engineering: Release Management Within Open Source Projects". Institute of Software Research, University of California, Irvine, [Online], 2003-03-03, haettu 2009-03-09. (Sivut 51–55.)
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.7020&rep=rep1&type=pdf#page=51

[35] VMware, "VMware Stage Manager 1.0.1 – User's guide". VMware, inc, Palo Alto, CA, [Online], 2008-09-10, haettu 2009-03-12. (Sivut 115–119, 123–125 ja 179–184.)
http://www.vmware.com/pdf/stagemanager101_Users_Guide.pdf

[36] Distributed Management Task Force, Inc., "Open Virtualization Format Specification". [Online], 2009-02-22, haettu 2009-03-19.
http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[37] VMware ja Xen Source, "The Open Virtual Machine Format Whitepaper for OVF Specification". [Online], 2007-09-07, haettu 2009-03-19. (Sivut 7–8.)
http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf

[38] C. Long, Cormac, "Network design principles". [Online], 2006-11-14, haettu 2009-03-22.
http://searchnetworkingchannel.techtarget.com/tip/0,289483,sid100_gci1229957,00.html

[39] P. Morrissey, "The Interactive Network Design Manual: How To Secure Your Network". Network Computing, [Online], 1996-11-15, haettu 2009-03-22.
http://www.networkcomputing.com/netdesign/security2.html

[40] VMware, "ESX 3.5 Basic System Administration Guide". VMware, inc, Palo Alto, CA, [Online], 2009-02-13, haettu 2009-03-23. (Sivut 83 ja 99.)
http://www.vmware.com/pdf/vi3_35/esx_3/r35/vi3_35_25_admin_guide.pdf

[41] Hard Disk Sentinel, "Can we believe in S.M.A.R.T?". [Online], 2009-04-13, haettu 2009-03-16.
http://www.hdsentinel.com/smart/

[42] P. Dorion, "The differences between incremental and differential backup", 2005-08-30:
http://searchstorage.techtarget.com/tip/0,289483,sid5_gci1119483,00.html

[43] Microsoft, "Operations guide: Backing up and restoring Virtual Server". Microsoft Technet, Virtual Server 2005, [Online], haettu 2009-03-11.
http://technet.microsoft.com/en-us/library/cc720377.aspx

[44] VMware, "Virtual Machine Backup Guide, ESX 3.5 & Virtual Center 2.5". VMware, inc, Palo Alto, CA, [Online], 2008-02-25, haettu 2009-03-11. (Sivut 28 ja 30–31.)
http://www.vmware.com/pdf/vi3_35/esx_3/r35/vi3_35_25_vm_backup.pdf

[45] A. J. Lewis, "LVM HOWTO". Rackable Systems, inc, [Online], 2006-11-27, haettu 2009-03-30. (Luku 13.4: Taking a Backup Using Snapshots.)
http://tldp.org/HOWTO/LVM-HOWTO/snapshots_backup.html

[46] J. Watkins, *Testing IT: An Off-the-Shelf Software Testing Process*, Cambridge University Press, 2000. ISBN: 0-521-79546-X. (Sivut 15–16.)

[47] J. Z. Gao, H.-S. J. Tsao ja Y. Wu, *Testing and Quality Assurance for Component-Based Software*, Artech House, Inc, 2003. ISBN: 1-58053-480-5. (Sivut 230–232.)

[48] Tom's hardware, "3.5 Desktop Hard Drive Charts, h2benchw 3.12: Max Write Throughput". Bestofmedia Group, [Online], 2009, haettu 2009-04-11.
http://www.tomshardware.com/charts/2009-3.5-desktop-hard-drive-charts/h2benchw-3.12-Max-Write-Throughput,1012.html

[www-1] Sun Developer Network, Java Enterprise Edition:
http://java.sun.com/javaee/

[www-2] Microsoft Application Platform:
http://www.microsoft.com/applicationplatform/default.mspx

[www-3] IBM CICS Family:
http://www-01.ibm.com/software/htp/cics/

[www-4] Common Object Request Broker Architecture:
http://www.corba.org/

[www-5] Oracle Weblogic Suite:
http://www.oracle.com/appserver/weblogic/weblogic-suite.html

[www-6] JBoss Community:
http://jboss.org/

[www-7] Apache Software Foundation, Tomcat:
http://tomcat.apache.org/

[www-8] Microsoft Windows Server 2003:
http://www.microsoft.com/windowsserver2003/default.mspx

[www-9] Microsoft Windows Server 2008:
http://www.microsoft.com/windowsserver2008/en/us/default.aspx

[www-10] IBM CICS Transaction Server:
http://www-01.ibm.com/software/htp/cics/tserver/

[www-11] Eclipse Open Development Platform:
http://www.eclipse.org/

[www-12] CruiseControl Continuous Integration Tool:
http://cruisecontrol.sourceforge.net/

[www-13] IBM Logical Partitioning:
http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/DZ9AR006/1.1?DT=19990630131355

[www-14] AMD Virtualization technology:
http://www.amd.com/virtualization

[www-15] IBM Advanced Power Virtualization:
http://www-03.ibm.com/systems/power/software/virtualization/

[www-16] Hitachi Virtage:
http://www.hitachi-america.us/products/business/ssg/products/bladesymphony_1000/virtage/index.html

[www-17] SUN Microsystems Hyper Privileged Execution Mode:
http://www.opensparc.net/opensparc-t1/index.html

[www-18] VMware ESX:
http://www.vmware.com/products/vi/esx/

[www-19] VMware Server:
http://www.vmware.com/products/server/

[www-20] Microsoft Virtual Server:
http://www.microsoft.com/windowsserversystem/virtualserver/

[www-21] Microsoft Hyper-V:
http://www.microsoft.com/windowsserver2008/en/us/hyperv.aspx

[www-22] Sun VirtualBox:
http://www.virtualbox.org/

[www-23] Sun xVM Family:
http://www.sun.com/software/products/xvm/index.jsp

[www-24] Kernel-based Virtual Machine:
http://kvm.qumranet.com/

[www-25] Xen Hypervisor:
http://www.xen.org/

[www-26] Oracle Virtual Machine:
http://www.oracle.com/technologies/virtualization/index.html

[www-27] Sun xVM Server:
http://www.sun.com/software/products/xvmserver/index.xml

[www-28] RedHat Virtualization Live Migration:
http://www.redhat.com/rhel/virtualization/solutions.html#
migration

[www-29] IBM General Parallel File System:
http://www-03.ibm.com/systems/clusters/software/gpfs/index.html

[www-30] RedHat Global File System:
http://www.redhat.com/gfs/

[www-31] VMware ESX 3.5 Hot Cloning:
http://jameshurst.us/2008/03/24/hot-cloning-with-esx-35

[www-32] VMware Open Virtual Machine Tools:
http://open-vm-tools.sourceforge.net/faq.php

[www-33] Nagios, monitoring tool:
http://www.nagios.org/

[www-34] Munin, monitoring tool:
http://munin.projects.linpro.no/

[www-35] Manifesto for Agile Software Development:
http://agilemanifesto.org/

[www-36] VMware Stage Manager:
http://www.vmware.com/products/sm/

[www-37] Distributed Management Task Force:
http://www.dmtf.org

[www-38] XenStore:
http://wiki.xensource.com/xenwiki/XenStoreReference

[www-39] Monit, monitoring tool:
http://mmonit.com/monit/

[www-40] Oracle Recovery Manager (rman):
http://www.oracle.com/technology/deploy/availability/htdocs/rman_overview.htm

[www-41] Sun Microsystems, Enterprise JavaBeans technology (EJB):
http://java.sun.com/products/ejb/

[www-42] W3C, Web Services Architecture:
http://www.w3.org/TR/ws-arch/

[www-43] Apache Proxy module:
http://httpd.apache.org/docs/2.2/mod/mod_proxy.html

[www-44] Apache Jakarta Jmeter:
http://jakarta.apache.org/jmeter/

[www-45] pi_css5-ohjelma:
http://myownlittleworld.com/miscellaneous/computers/piprogram.html

[www-46] GNU Coreutils, dd:
http://www.gnu.org/software/coreutils/manual/html_node/dd-invocation.html

Liite A

Verkko- ja palvelinrakenne virtualisoinnin jälkeen.